Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Estudio comparativo entre metodologías tradicionales y metodologías ágiles aplicadas a proyectos IT en entorno industrial

Trabajo fin de Máster

MASTER INTERUNIVERSITARIO EN DIRECCIÓN DE PROYECTOS

Autor: Iñaki Torres Valencia

Directoras: Amaia Pérez y Sara Marcelino

Pamplona, noviembre de 2020

Resumen

Se pretende comparar diferentes metodologías de gestión de proyectos tradicionales con los nuevos enfoques que proponen las metodologías ágiles y ver su posible aplicación en la gestión de un proyecto IT dentro de un entorno industrial.
Estas metodologías ágiles suelen ser relacionadas con el desarrollo de software, donde está más extendida su aplicación y sus ventajas.

El objetivo es analizar la conveniencia de su aplicación a proyectos no puramente de desarrollo de software, como pueden ser los llamados “proyectos de Industria 4.0” que buscan la incorporación de tecnologías de la información a los procesos de producción.

Palabras clave: *Industria 4.0, Proyectos IT, metodologías tradicionales, metodologías ágiles, gestión proyectos, SCRUM, Kanban*

Indice

GLOSARIO

Agile Coach	Persona formada en metodología Agile, cuyo objetivo es velar porque se desarrolle la metodología.
Burndow Chart	Gráfico de quemado, en el sentido de cuánto se lleva gastado o realizado en el proyecto, comparado con el ritmo teórico.
Pain-Points	Puntos del proceso que identifican aspectos de mejora.
Planning Poker	Proceso de identificación de la duración de las tareas mediante cartas.
Product Owner	Representa la voz del cliente: conoce las necesidades como negocio, tiene la visión global del *product*, y es capaz de tomar decisiones.
Product Backlog	Lista de funcionalidades requerida por negocio. Se encuentra priorizada.
Retrospective Meeting	Reunión tras la realización de un sprint, para encontrar fortalezas y mejoras.
Scrum	*Marco de trabajo* para gestionar proyectos dentro de la metodología Agile.
Scrum Master	Rol con capacidad de gestión. En Scrum, hace referencia a la persona que se encarga de que apliquen las metodologías seleccionadas, y resuelve cualquier problema acerca de las mismas durante todas las etapas del sprint.
Sprint	Ciclo de trabajo. Proceso organizado en un período de tiempo (usualmente de una a cuatro semanas) que se repite. Es una forma de organización utilizada en Scrum.
Sprint Backlog	Conjunto listado de tareas que conforman un sprint.
Sprint Daily	Reunión para revisar la planificación del día anterior, y el día en curso.
Sprint Review	Reunión de revisión de entregrables con el Product Owner.
Sprint Planning	Reunión de planificación de las tareas del Product Backlog.
Time-Box	Duración de un evento (reunión, sprint, etc.).
ACM	Association for Computing Machinery
AEIPRO	Asociación Española de Ingeniería de Proyectos
AENOR	Asociación Española de Normalización y Certificación
ANSI	American National Standards Institute
APM	Association for Project Management
APMBOK	Association for Project Management Body Of Knowledge
CMMI	Capability Maturity Model Integration

DSDM	Dynamic Systems Development Method FDD
FDD	Feature Drive Development
ICB	IPMA Competence Baseline
IEEE	Institute of Electrical and Electronics Engineers
IPMA	International Project Management Association
IT	Information Technology
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
PMBOK	Project Management Body Of Knowledge
PMP	Project Management Professional
PMI	Project Management Institute
PMO	Project Management Office
PRINCE2	Projects in a Controlled Environment
PROMPT	Project Resource Organisation Management Planning Technique
RUP	Rapid Application Development
SEI	Software Engineering Institute
SWEBOK	Software Engineering Body of Knowledge
TDD	Test Driven Development
TIC	Tecnologías de la Información y Comunicación
UML	Unified Modeling Language
XP	eXtreme Programming

1 Introducción

1.1. Estructura del Trabajo Fin de Máster

Cap. 1**: Introducción.**

Comienza por introducir la temática explicando qué es un proyecto, en particular los de tipo software o IT y cuáles son las causas de su fracaso. Además, describe el enfoque que tiene las metodologías tradicionales y ágiles. También se explica las motivaciones, objetivos y fuentes de información que se han utilizado para la realización de este trabajo.

Cap. 2: **Proyectos IT en la Industria 4.0.**

Se hace una presentación del término "Industria 4.0" una evolución histórica del campo así como se definen términos importantes que se repiten y los modelos de ciclo de vida de los proyectos IT en un entorno industrial. Además, se explica las particularidades de los proyectos en la llamada industria 4.0.

Cap. 3: **Metodologías**.

En esta sección se hace una introducción sobre las metodologías de gestión de proyectos, enfocadas a los proyectos IT y por tanto al desarrollo del software: Desde una visión histórica cómo surgen, en qué consisten, y cuáles son los principios en los que se basan, así como las ventajas y desventajas respecto a otras metodologías de desarrollo de proyectos.

Se presentan algunas de las metodologías más importantes pertenecientes a las diferentes familias: Tradicionales, ágiles e híbridas.

Cap. 4: **Ejemplo de metodologías ágiles**

En esta sección se hace una introducción sobre el sistema Kanban, y su uso en el desarrollo de proyectos de IT, para continuar abordando la metodología Scrum: su origen, elementos principales de la metodología y herramientas relacionadas con el uso de dicha metodología.

Estos son el sistema y la metodología que el autor conoce con más detalle en su experiencia laboral y por ello hace un estudio más detallado.

Cap. 5: **Comparativa entre metodologías.**

Se pretende comparar las diferentes familias de metodologías: las tradicionales y las ágiles.

Primero desde un punto de vista general y finalmente comparando dos de las más representativas de cada rama: PMBOK y SCRUM

Cap. 6: **Conclusiones finales**.

Conclusiones del trabajo y próximas áreas de desarrollo.

Anexos: Manifiesto Ágil. Transcripción del manifiesto que está alojado en su página web.

1.2. Gestión de proyectos

Hay muchas definiciones de lo que es un proyecto y su gestión por ser unos términos generales que tienen bastantes ámbitos de actuación. Dos posibles definiciones serían las siguientes.

- Un proyecto es un esfuerzo temporal encaminado a la creación de un producto, servicio o resultado único.
- La gestión de proyectos es aplicar tanto la ciencia como el arte para planificar, organizar, poner en marcha, dirigir y controlar el trabajo de un proyecto para cumplir con los objetivos y metas de la organización.

Una gestión eficiente de proyectos es muy importante para las organizaciones por diversos motivos como la alta complejidad de los proyectos, la fuerte competencia entre empresas, la reducción de costes e ineficiencias, la gestión de equipos humanos, asegurar la calidad, la diversidad de involucrados en los proyectos, el control de la desviación según la planificación y un largo etcétera.

Dentro de una organización (figura 1) un grupo de proyectos pueden formar un programa (línea de negocio) y estos a su vez un portfolio (área de negocio). Se pretende estudiar algunos aspectos de la gestión de un tipo de proyectos (IT, software o industria 4.0), sin entrar en la gestión de programas y portfolios.

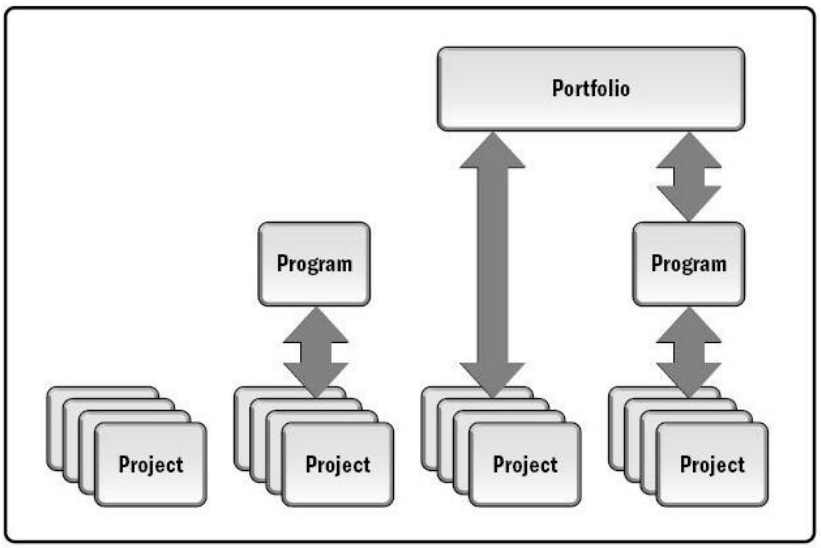

Figura 1. Relación entre portfolios, programas y proyectos

1.3. Qué son los proyectos IT

Como se ha dicho en el apartado anterior, se van a considerar principalmente los proyectos de tipo IT o desarrollo de software, con sus particularidades.

Básicamente llamamos proyectos IT (proyectos de tecnologías de información) a los proyectos con un componente tecnológico principal, y aunque en su mayoría hay una importante carga de desarrollo de software, conviene distinguirlos de un proyecto puramente de desarrollo de software, como puede ser el desarrollo de una aplicación o programa informático, de lo que es un proyecto IT.

La gestión de proyectos de IT incluye la supervisión de los proyectos de desarrollo de software, instalaciones de hardware, actualizaciones de red, despliegues de computación en nube y virtualización, proyectos de gestión de datos y análisis de negocios, y la implementación de servicios de TI.

En el entorno industrial, cada vez es más difícil que haya un proyecto que no pueda ser considerado como un proyecto IT tal y como se ha descrito anteriormente.

Con la digitalización de los procesos, pocas áreas de la producción quedan fuera de un entorno IT ya que se busca cada vez más la automatización de procesos, la intercomunicación entre las máquinas (IoT), la visualización de datos en tiempo real, el mantenimiento predictivo…

Como se ha explicado, el desarrollo de software es un componente principal en los proyectos IT. El término software fue usado por primera por *John W. Tukey* en un artículo académico en la American Mathematical Monthly en 1958. Se podrían dar varias definiciones de qué es software.

- Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora (Real Academia de la Lengua).
- Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación (IEEE).
- Instrucciones (código de ordenador) que cuando son ejecutadas proveen las deseadas características, funciones y rendimiento; estructuras de datos que permiten a los programas manipular información e información descriptiva para la operación y uso de los programas (*Pressman*, 2010, [12]).

El término Ingeniería de Software fue sugerido en las conferencias organizadas por la North Atlantic Treaty Organisation (NATO) en 1968 y 1969 para discutir la llamada “crisis del software”. Se le llamó así por las dificultades en desarrollar grandes y complejos sistemas en la década de 1960 porque no se obtenían los resultados deseados, había sobrecostes y tenían poca flexibilidad. Se propuso que la adopción de un enfoque ingenieril al desarrollo de software debería reducir los costes de desarrollo y conseguir un software más seguro. Han pasado casi 50 años y todavía los proyectos software tienen altos porcentajes de fracaso (se estudiará en los apartados posteriores) a pesar de la enorme evolución que ha tenido el sector: constante innovación en tecnologías software y hardware, herramientas de desarrollo de software y de gestión, documentación libre y gratuita en Internet, certificaciones, estándares, metodologías, etc...

Podemos llegar a la conclusión de que realizar un proyecto desde el punto de visto técnico y de gestión ha sido y sigue siendo complejo.

Para ayudar a entender que es la ingeniería del software, se enuncian algunas definiciones en orden cronológico.

- Trata sobre el establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales (*Bauer*, 1972).
- La aplicación práctica del conocimiento científico al diseño y construcción de programas de computador y a la documentación asociada requerida para desarrollarlos, operarlos y mantenerlos. Se conoce también como desarrollo de software o producción de software (*Bohem*, 1976).
- El estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software (*Zelkovitz*, 1978).
- La aplicación sistemática, disciplinada y cuantificable al desarrollo, operación y mantenimiento del software (ISO/IEC/IEEE, 1993).
- Es la disciplina ingenieril concerniente a todos los aspectos de la producción software: especificaciones, desarrollo, validación y evolución (*Sommerville*, 2011, [13]).
- Es la ingeniería encargada de desarrollar sistemas intensivos en software, esto es, una colección de hardware, software y manuales. El software es el principal componente que integra y coordina la operación del sistema (Project Management Body Of Knowledge (PMBOK), 2016, [1]).

Estas definiciones se han ido ampliando cada vez más a lo largo de los años y evolucionando para adaptarse a la realidad. La creencia de que el software es tan sólo el código que se ejecuta en una máquina se ha quedado obsoleta ya que, cualquier sistema software necesita una buena documentación para poder comprender su diseño, su arquitectura y los requisitos que satisface para poder operar, mantener y realizar evolutivos en el futuro.

En concreto, en los proyectos IT en el entorno de la industria, el código es solo una parte de los desarrollos y creaciones necesarias para obtener el producto final.

Normalmente, no se trata de un solo programa si no de muchas pequeñas aplicaciones y funciones, programados en diferentes lenguajes y con diferentes propósitos que combinados ayudan a alcanzar el fin deseado.

Si prestamos atención a las propiedades clásicas del código, Sommerville [13] enumera las siguientes características que definen a un buen software.

Mantenibilidad: el código debe estar escrito de tal manera que pueda evolucionar para satisfacer las necesidades cambiantes del cliente. Esto es un atributo crítico porque los cambios son inevitables en un entorno empresarial cambiante.

Fiabilidad y seguridad: la fiabilidad del software incluye las características de confiabilidad, seguridad y protección. El software confiable no debería causar daños físicos o económicos en caso de fallo. Los usuarios malintencionados no deberían ser capaces de acceder o dañar el sistema.

Eficiencia: el software no debe hacer un uso excesivo de recursos del sistema. Por lo tanto, la eficiencia incluye la capacidad de respuesta, el tiempo de procesamiento, la utilización de la memoria, etc...

Aceptabilidad: el software debe ser aceptado por los usuarios para el que fue diseñado. Esto significa que debe ser comprensible, útil y compatible con otros sistemas que utilizan.

Estos conceptos siguen siendo válidos a día de hoy y más aún en un tipo de proyecto donde se involucran muchas partes, personas, tecnologías, máquinas

1.4. Motivación, objetivos y fuentes de información

Este Trabajo Fin de Máster (TFM) tiene como objetivo presentar y comparar las distintas familias de metodologías de dirección de proyectos dentro del mundo de los proyectos IT en industria.

Este tipo de proyectos tiene particularidades propias que lo distinguen tanto del resto de proyectos puros de ingeniería como de los que tradicionalmente conocemos como un proyecto de ingeniería del software, combinando procesos y maneras de ambos entornos. Consecuentemente, la forma de abordar la dirección y gestión de los proyectos debería ser distinta, intentando mejorar la gestión de aquellos puntos donde tradicionalmente los proyectos fracasan.

Algunas particularidades relevantes son:

- El principal recurso para realizar el proyecto son los ingenieros y entre ellos, tienen un papel fundamental los ingenieros de software. Por tanto, gran parte la actividad del proyecto recae en las personas no pudiendo sistematizar o automatizar gran parte de los trabajos.
- Generalmente los proyectos empiezan con unas especificaciones vagas porque el cliente no sabe especificar en detalle los requisitos y es a medida que avanza el proyecto cuando va haciendo modificaciones o ampliaciones.
- Hay que definir métricas propias para evaluar y controlar el desarrollo del proyecto por ser una actividad intelectual realizada por personas. La tarea de supervisión es compleja
- La Ing. Software como la conocemos hoy en día y el desarrollo de aplicaciones son relativamente nuevos y no hay tanta experiencia en la gestión y dirección de proyectos como otras ingenierías con más recorrido e historia.

Por éstas y otras razones han surgido diversas metodologías que tratan de estandarizar la gestión y dirección de proyectos software. Las más importantes son:

- Project Management Body Of Knowledge (PMBOK) es una guía basada en procesos realizada por el Project Management Institute (PMI). Es la más utilizada universalmente en el ámbito de la ingeniería y en 2013 sacaron una extensión de su guía para proyectos software [2].
- ISO 21500 [6] es la norma de gestión de proyecto de International Organization for Standardization (ISO) siendo prácticamente igual a PMBOK.

- IPMA Competence Baseline (ICB) es una guía basada en competencias realizada por el International Project Management Association (IPMA) [3] .
- Projects in a Controlled Environment (PRINCE2) es una guía realizada por el UK Office of Government Commerce (OGC) [4].
- Metodologías ágiles. Es un relativamente nuevo paradigma de desarrollo software con múltiples técnicas para llevarlo a la práctica.

1.5. El fracaso de los proyectos IT en entorno industrial

Uno de los componentes principales de los proyectos IT es el desarrollo de software. Existe mucha literatura acerca de los problemas en lo proyectos de software y es por ello que podemos asimilar los problemas de los proyectos IT al fracaso de proyectos software.

Este fenómeno no es algo reciente, ya en 1968 hubo la llamada "*crisis del software*". Se constató una serie de sucesos que se venían observando en los proyectos de desarrollo de software:

- No terminaban en plazo.
- No se ajustaban al presupuesto inicial.
- Baja calidad del software generado.
- Software que no cumplía las especificaciones.
- Código imposible de mantener que dificultaba la gestión y evolución del proyecto.

Si nos fijamos en lo que ocurre en las empresas, en muchos casos los proyectos IT fallan por problemas relacionados con:

Tiempo y costos: Los proyectos se completan con mucho retraso, por lo general los costos llegan a niveles muy por encima de los estimados
Requerimientos: Los proyectos que logran finalizar a tiempo no satisfacen los requerimientos de los usuarios
Planificación: Los proyectos que terminan de forma exitosa lo hacen a costa de generar un gran estrés en el personal técnico y los usuarios y de haber requerido gran cantidad de horas de sobre tiempo.

El estudio más citado y relevante es el *The CHAOS Report* publicado por el *Standish Group*. Este grupo se creó en 1985 por una serie de profesionales de West Yarmouth (Massachussets) para obtener información de los proyectos software fallidos e intentar encontrar las causas de los fracasos.
Su primer informe fue en 1994 y con el tiempo se ha convertido en un referente sobre los factores que inciden en el éxito o fracaso de los proyectos.
Este grupo clasifica los proyectos en tres tipos:

- **Successful** (éxitoso). Se completa en costes, plazos y tiene todas las funcionalidades. El informe deja fuera aspectos discutibles como la calidad, el riesgo y la satisfacción del cliente.
- **Challenged** (discutido). Se completa y es operacional pero no ha cumplido en costes, plazos y/o funcionalidades.
- **Failed** (fallido). Es cancelado antes de completarse.

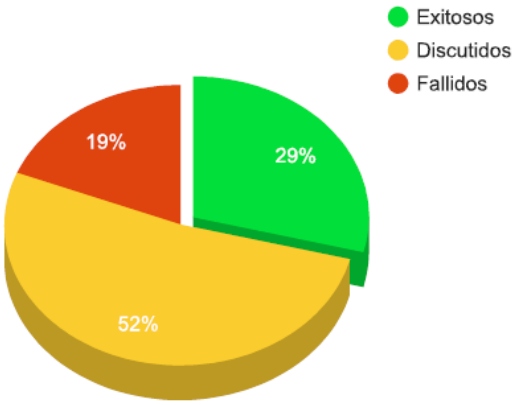

Figura 2. Estado de los proyectos a su finalización en el año 2015. Dato de CHAOS Manifesto

En la figura se muestra el estado de los proyectos a su finalización en el año 2015. Sorprende el bajo porcentaje de proyectos exitosos debido a que el número de proyectos discutidos es muy alto. Si no hubiese proyectos cuestionados, habría un 81% de exitosos. Es decir, con una buena labor de gestión del proyecto, no habría tantos proyectos discutidos.

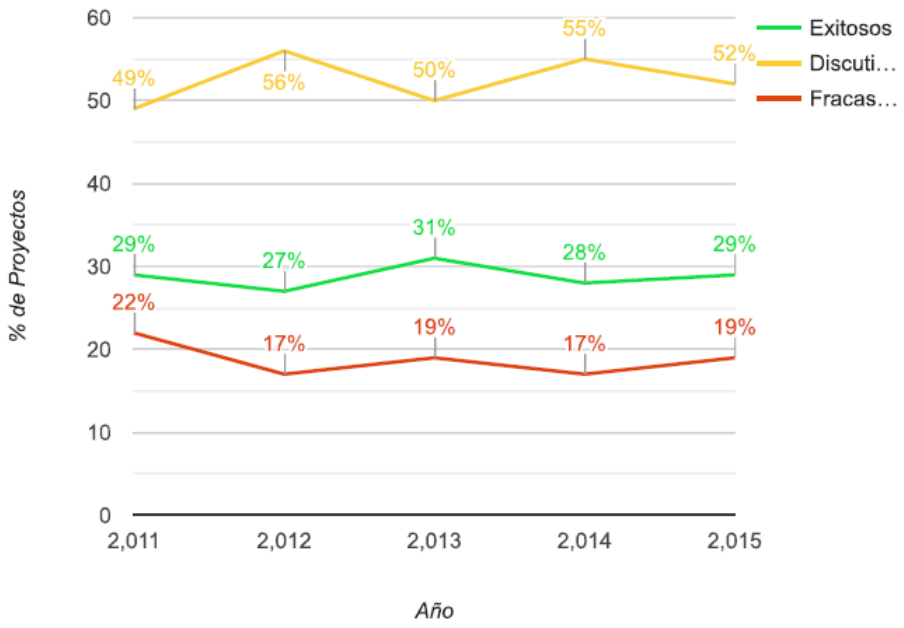

Figura 3. Evolución histórica (2011-2015) de los estado de los proyectos a su finalización. Dato de CHAOS Manifesto 2015.

La anterior tabla expone la evolución del porcentaje de éxito de los proyectos del año 2011 al 2015. Se observa que el porcentaje de fracaso es prácticamente constante.

Parece ser que a pesar de las nuevas guías y estudios sobre la gestión de proyectos, no hay una clara mejoría en los resultados de los mismos. Es por eso que debemos seguir investigando e insistiendo en este campo

2 Proyectos IT en Industria 4.0

Este trabajo se centra en los Proyecto IT en un entorno industrial, por lo que resulta necesario explicar el momento en el que nos encontramos actualmente en el mundo de la industria, que en los últimos años ha sido enmarcado en el concepto de "Cuarta revolución industrial" o más comúnmente llamado "Industria 4.0"

2.1 Industria 4.0

Repasando brevemente la historia, podemos hablar de la **primera revolución industrial** como aquella que surgió en 1784 con el primer sistema mecanizado implantado en las máquinas de vapor.

La **segunda revolución** apareció en 1870 cuando se inventó la primera cinta transportadora que facilitó la producción en serie.

Posteriormente, en 1969 surgieron los controladores programables, que permitían automatizar de forma electrónica la producción. Aquí podemos fechar la **tercera revolución**.

El concepto de **Industria 4.0** fue mencionado por primera vez en la feria de Hannover (feria dedicada a la tecnología industrial) de 2011 con la intención de poner en marcha un proyecto que llevara a cabo la concepción y desarrollo de la fábrica inteligente asociada a la cuarta revolución industrial, una visión de la fabricación informatizada con todos sus procesos interconectados entre sí haciendo uso del internet de las cosas (IoT), hoy en día llamado el internet industrial de las cosas (IIoT).

Dos años más tarde el gobierno alemán puso en marcha esta iniciativa con la idea de hacer frente a los grandes avances en materia industrial que se estaban llevando a cabo en los países emergentes como por ejemplo China, y que, al no poder competir en costes de producción, la idea era poder superarlos en tecnología industrial y en la capacidad de fabricar productos de manera individualizada.

Aunque como se ha podido ver, estos países no han tardado en querer subir al carro de esta **cuarta revolución industrial** una vez puesta en marcha en Europa y en Estados Unidos, donde por cierto el concepto de **Industria 4.0 o** Industrial Intelligence a diferencia de en Europa, es llamado **Smart Manufacturing** o **Industrial Internet**.

El término de **Industria 4.0** abarca muchos conceptos y finalidades, pero los primeros avances en este ámbito implican la incorporación de una mayor flexibilidad e individualización de los procesos de fabricación. La industria automotriz es pionera en la necesidad de poner en marcha estos procesos de fabricación flexibles e individualizados, y es donde ya se están viendo grandes avances en este ámbito debido a que los fabricantes tienen que adaptar los vehículos a las necesidades individuales de los clientes de manera rápida y eficiente.

2.1.1 Principios de diseño

Los principios de diseño en que se basan las empresas y organizaciones para implementar la **Industria 4.0** son los siguientes:

• **Decisiones descentralizadas.** Los **Sistemas Ciberfísicos** tienen que ser capaces de tomar decisiones por sí mismos y de realizar sus labores de la manera más autónoma posible.

• **La interoperabilidad.** Las personas, aparatos, sensores y máquinas tienen que ser capaces de comunicarse entre sí. Aquí juega un papel muy importante el **Internet de las Cosas** (IOT).

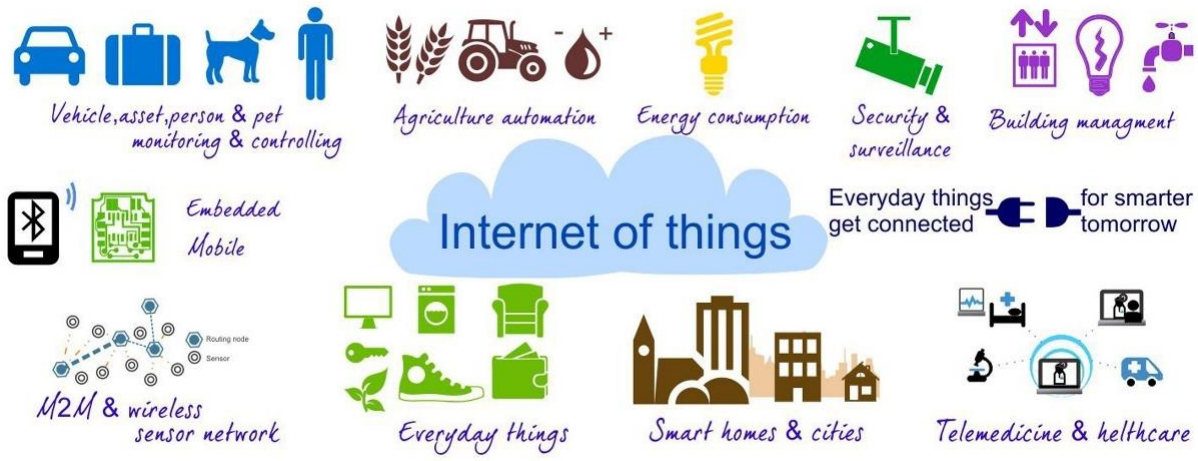

Figura 4. Imagen de datasciencebe.com

• **Asistencia técnica.** Los **Sistemas Ciberfísicos** tienen que ofrecer soporte y colaboración a los humanos en todas aquellas tareas que sean peligrosas, produzcan fatiga, cansancio o sean desagradables. Y por otro lado tienen que ofrecer una ayuda o soporte que sea capaz de añadir información visual inteligible para el humano de manera que pueda avanzarse a los posibles problemas y/o resolverlos en el menor tiempo posible.

• **Transparencia informativa.** Los sistemas de información tienen que ser capaces de crear una copia virtual o "**Digital Twin**" del mundo físico que les rodea a través de los datos recopilados a través de sus sensores y otros dispositivos conectados en su ecosistema.

2.2 Retos de la Industria 4.0

Se espera que el nuevo concepto de **Industria 4.0** asociado a la **cuarta revolución industrial** y a los Sistemas Ciberfísicos, sea capaz de impulsar cambios fundamentales al mismo nivel que las tres revoluciones precedentes. Recordemos que en la primera revolución se introdujo la mecánica movida por la energía generada por el agua y el vapor, en la segunda se introdujo la electricidad y la producción en masa inventada por Henry Ford, y la tercera la automatización y la proliferación de las tecnologías de la información.

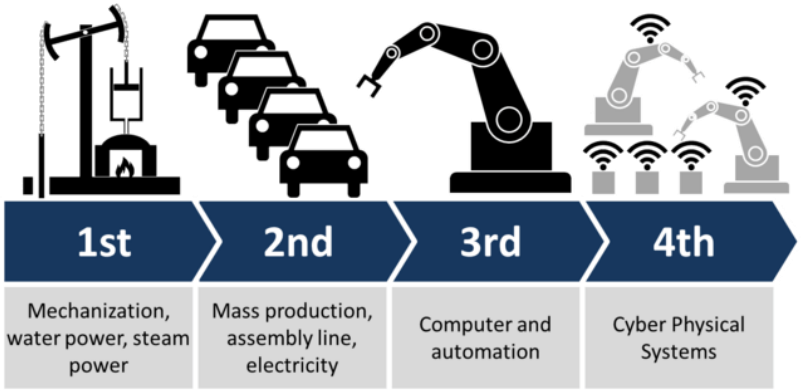

Figura 5. Imagen de: Christoph Roser at AllAboutLean.com, CC BY-SA 4.0

Si tuviéramos que definir cuáles son los retos más importantes a los que se van a tener que enfrentar las empresas y organizaciones que implementen la **Industria 4.0** serían los siguientes:

El **desarrollo de software y sistemas de análisis** que convierten la inmensa cantidad de datos producidos por las **fábricas inteligentes** en información útil y valiosa.

Figura 6. Imagen de: Christoph Scholz, CC BY-SA 2.0

Responder a la problemática actual en **ahorro de energía** y en **gestión de recursos naturales.**

Figura 7. Imagen de: 3M Smart Energy

La **Ciberseguridad**. Los sistemas tienen que estar fuertemente protegidos ante cualquier Ciberataque debido a la necesidad de las industrias a introducir sistemas open source para adecuarse a la **transformación digital** de sus procesos, es por ello que la **Ciberseguridad** es un aspecto clave en esta **cuarta revolución industrial**.

Figura 8. Imagen de: Christoph Scholz, CC BY-SA 2.0

Evitar fallos informáticos o de comunicación debidos a la gran inmersión de software e infraestructura IT a la que tienen que someterse los sistemas informáticos de las empresas. Estos fallos podrían producir grandes pérdidas económicas.

La **falta de personal cualificado** para conducir a sus empresas hacia la **cuarta revolución industrial**.

Reticencias de los directivos de las empresas a emprender el camino hacia la **transformación digital** de sus negocios.

Ciertas empresas (sobre todo las PYMES) deberán empezar por **optimizar sus procesos** o bien adecuar antes sus infraestructuras y/o instalaciones productivas (industria 3.0) antes de emprender el camino hacia la **industria 4.0**.

Pérdida masiva de puestos de trabajo debido a la automatización de procesos donde va a tener que producirse un cambio en las tareas que desempeña hoy en día el operario de líneas de fabricación. La inmensa mayoría van a tener que reconvertirse profesionalmente, adecuándose a nuevos trabajos que a día de hoy no existen y que surgirán con el cambio de paradigma.

Asegurar el **retorno de la inversión** (ROI) generada por la nueva tecnología requerida para la transformación digital de las empresas.

A parte de los puntos mencionados anteriormente, cabe destacar que actualmente la gran mayoría de empresas no disponen de unas infraestructuras a la última en tecnología y como consecuencia van a tener más problemas y van a requerir de mayor inversión para alcanzar la transformación digital de sus procesos productivos, ya que para ello son necesarios unos sistemas de automatización industrial de última generación que puedan integrar cada vez más sensores, nuevas tecnologías, y que tengan capacidad de comunicación inalámbrica.

Las fábricas deberán ganar en capacidad de interoperabilidad y en recopilar datos de forma masiva de los elementos que compongan sus procesos de producción para lograr mejoras reales en la eficiencia de fabricación y flexibilidad, y a su vez deben ser capaces de gestionar y analizar estas grandes cantidades de datos.

Los ingenieros que están inmersos en este ámbito, llevan ya unos años desarrollando soluciones industriales para el análisis de datos en el área del **Big Data** y **Data Analytics** basadas en open source, aprovechando las ventajas de este tipo de plataformas.

Las empresas deben implementar tecnologías y software **Big Data** capaz de dar salida a grandes cantidades de datos recogidos del entorno de fabricación para poder analizarlos y realizar las acciones necesarias en cada caso.

Lo que está claro es que tanto las pequeñas como las medianas o grandes empresas se tienen que ir concienciando de que la transformación digital de sus plantas de producción y otros ámbitos de sus negocios, es necesaria si no se quieren quedar atrás en la carrera hacia la cuarta revolución industrial que no ha hecho más que empezar y a la que le queda un largo recorrido todavía.

Cabe destacar que el factor más importante en el camino hacia la **industria 4.0** son las personas, puesto que, si no somos realmente conscientes de que este cambio de paradigma es necesario, el éxito de su implementación no será posible.

2.3 Objetivos de la Industria 4.0

La Industria 4.0 implica la promesa de una nueva revolución que combina técnicas avanzadas de producción y operaciones con tecnologías inteligentes que se integrarán en las organizaciones, las personas y los activos.

Esta revolución está marcada por la aparición de nuevas tecnologías como la robótica, la analítica, la inteligencia artificial, las tecnologías cognitivas, la nanotecnología y el Internet of Things (IoT), entre otros. Las organizaciones deben identificar las tecnologías que mejor satisfacen sus necesidades para invertir en ellas. Si las empresas no comprenden los cambios y oportunidades que trae consigo la Industria 4.0, corren el riesgo de perder cuota de mercado.

Tradicionalmente, acostumbrados a los datos y las comunicaciones lineales, el cambio que supone esta nueva revolución industrial -proporcionando acceso en tiempo real a los datos y la inteligencia de negocio- transformará la forma en que llevan a cabo sus negocios. La integración digital de la información desde diferentes fuentes y localizaciones permite llevar a cabo negocios en un ciclo continuo. A lo largo de este ciclo, el acceso en tiempo real a la información está impulsado por el continuo y cíclico flujo de información y acciones entre los mundos físicos y digitales. Este flujo tiene lugar a través de una serie de pasos iterativos conocido como PDP –por sus siglas en inglés physical-to-digital-to-physical-:

- Del mundo físico al digital. Se captura la información del mundo físico y se crea un registro digital de la misma.
- De digital a digital. En este paso, la información se comparte y se interpreta utilizando analítica avanzada, análisis de escenarios e inteligencia artificial para descubrir información relevante.
- Del mundo digital al físico. Se aplican algoritmos para traducir las decisiones del mundo digital a datos efectivos, estimulando acciones y cambios en el mundo físico.

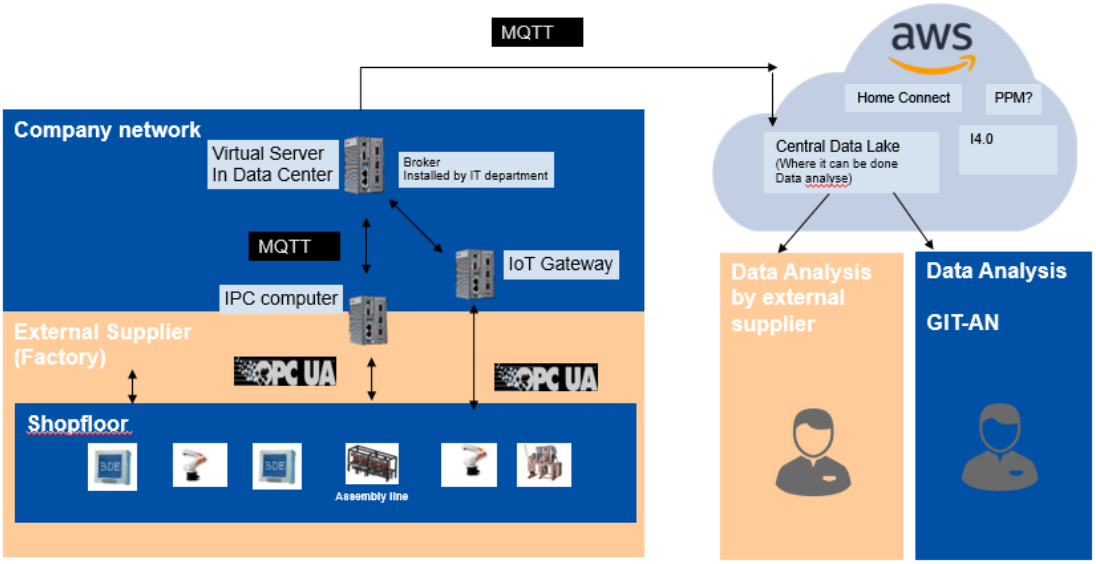

Figura 9. Esquema de proyecto IT. Departamento I4.0

2.4 Gestión de Proyectos IT

A lo largo del trabajo se enumerarán diferentes metodologías y procesos para lograr una buena gestión de proyectos IT en el entorno de industria 4.0.

A modo de resumen podemos resumir que una buena gestión de proyectos IT incluye:

- **Supervisión de proyectos de desarrollo de software.** Para ello veremos que la tendencia en los últimos años son las metodologías ágiles
- **Instalaciones de hardware.** Uno de los componentes principales de lo que es un proyecto de Industria 4.0 y también una de las partidas de presupuesto principal que hay que tener en cuenta a la hora de gestionar los recursos
- **Actualizaciones de red.**
- **Computación en la nube y virtualización.**
- **Gestión de datos.**
- **Análisis de negocios.**
- **Implementar servicios de tecnologías de la información.**

Como podemos ver, el factor tecnología está muy presente en el desarrollo de un proyecto IT.
Por ello, es frecuente que durante la gestión de un proyecto IT un avance de la tecnología de forma imprevista nos cause un problema, así que hay que estar preparados para solventarlo sobre la marcha.

3 Metodologías

En esta sección haremos una introducción sobre las metodologías de proyectos, enfocadas en los proyectos IT con un componente principal de desarrollo del software: trataremos desde una visión histórica cómo surgen, en qué consisten, y cuáles son los principios en los que se basan, así como las ventajas y desventajas respecto a otras metodologías de desarrollo de proyectos.

Posteriormente, trataremos dentro del marco de las metodologías ágiles, la metodología SCRUM que será la que se usará en el modelo con el que se desarrollará este trabajo.

3.1 Definición de Metodología

Según la definición realizada por Avison y Fitzgerald en el libro "*Desarrollo de Sistemas Informacionales. Metodologías, Técnicas y herramientas*":

"Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Una metodología está formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo." (Avison y Fitzgerald 1998)

Esta definición de metodología hace referencia a varios componentes de esta: las fases que componen las metodologías, las herramientas que se utilizan en el uso de estas, y las técnicas de las que se disponen para aplicar las metodologías.

Sin embargo, nos podemos seguir planteando que una metodología, debe ir más allá de ser una simple recolección de una serie de pasos a realizar o seguir, para conseguir algo.

Mediante esta definición de los autores, podemos ver que las diferencias presentes en las distintas metodologías están basadas en las distintas técnicas, y por tanto herramientas, aplicadas en las distintas fases, sobre el mismo contenido de estas.

3.1.1 Terminología de las metodologías

Según la R.A.E. una **técnica** se entiende como *"un conjunto de procedimientos y recursos de los cuales se sirve una ciencia o un arte".*
Para el caso en estudio, aplicado a la metodología, los procedimientos consistirán en unas normas específicas con las que realizar una serie acciones que compondrán el proceso de desarrollo (generalmente en más de una fase), y los recursos serán habitualmente las herramientas para llevar a cabo dichos procedimientos.
Podemos encontrar, que el término ***técnica***, se utiliza para hacer referencia a una metodología, pero esto sería un error, puesto que una metodología puede utilizar diferentes técnicas y no es en sí misma una sola técnica.

Una **herramienta**, según la *R.A.E.,* se define como *"un conjunto de instrumentos que se utilizan para desempeñar un oficio o un trabajo determinado*". En nuestro caso, las herramientas constituirán los medios a través de los cuales podremos llevar a cabo las técnicas descritas por las distintas metodologías.

Este instrumento, aplicado a las condiciones de contorno de este proyecto, consistiría en un producto informático. Un ejemplo de herramienta para desarrollar aplicaciones informáticas podría ser una herramienta de modelado, repositorios de software, compiladores, depuradores, conversores de códigos, etc.

Un **método**, según la *R.A.E.*, es *"el modo de obrar o proceder, hábito o costumbre que cada uno tiene y observa"*. Este método, nos dirá dónde están los límites: nos guiará a lo largo de una tarea. A cada técnica, comentada anteriormente, se le aplica un método, que nos ayuda a usar las técnicas mediante las herramientas concretas.

Un **proceso de desarrollo**, en este caso, **de software**, se podría definir como el grupo de tareas que, son necesarias realizar, para crear un producto, en este caso informático. En este proceso, que se puede hacer de muchas maneras y con resultados, totalmente diferentes, es donde las metodologías tratan de dar unas pautas comunes, para obtener un buen software. Dicho proceso, está compuesto en sí mismo por más procesos, pero a su vez, también está conformado por subprocesos, y es en el control de dichos procesos, donde aparece de forma implícita la calidad.
Gran parte de las metodologías, introducen como una técnica más dentro del proceso, el control de los procesos con el fin de garantizar una solución software de calidad. Habitualmente, este control es llevado a cabo como si se tratase de un producto más, y se certifican para dar fe de que se está siguiendo el proceso conforme a la norma.

Un modelo de procesos es una representación esquemática de las distintas etapas y actividades, que se deben realizar para obtener un proceso. Los modelos de procesos que hay a día de hoy son (Derniame, Kaba y Wastell 1999):

- **Modelo secuencial:** Cuyo máximo exponente conocido es la metodología **Waterfall**.
  Se parte de una recepción y análisis, de los requerimientos de los clientes. Tras dicho análisis, se establece un conjunto de requisitos funcionales y no funcionales necesarios, que serán documentados para que, en la fase de diseño, los ingenieros trabajen entre sí de cara a establecer una arquitectura sobre la que se desarrollará el software. En la siguiente etapa, los desarrolladores generan dichas funcionalidades, y por último, el sistema es probado y validado por los usuarios y/o clientes, y entregado, o puesto en producción.
  El modelo *waterfall* no resuelve las modificaciones de los requisitos, ya que no permite cambios una vez avanzado en las fases del proyecto, porque supondría empezar de nuevo.

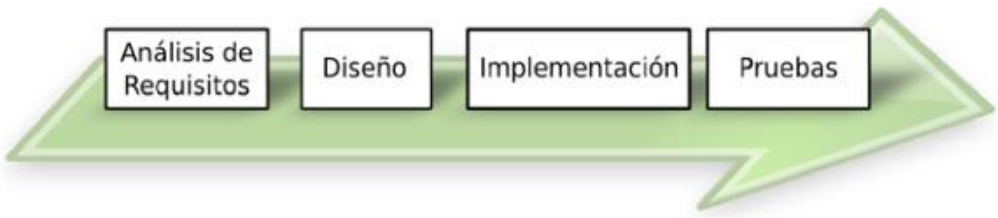

Figura 10. Modelo secuencial.

- **Desarrollo incremental:** disminuyen el tiempo total partiendo el proyecto en tramos concatenados. De la misma manera que en *waterfall*, los requerimientos se analizan antes de forma previa. La diferencia está, en que cada requerimiento se desarrolla de forma

independiente. De esta forma, el desarrollo de cada parte se puede solapar, ahorrando tiempo gracias al reparto de las tareas a distintos equipos.

- **Desarrollo iterativo**: se centra en entender bien los requerimientos que puedan variar y la gestión de riesgos. En este modelo, el proyecto se divide en varias iteraciones, de forma que en la última se obtenga un producto final. La primera iteración viene a ser como un boceto, al que se le van añadiendo funcionalidades en cada repetición de ciclo. En cada una de ellas, se utiliza el modelo secuencial. Funciona en entornos variables, debido a que los requerimientos no cambian en cada ciclo

- **Modelo en Espiral**: mezcla lo mejor del modelo tradicional, introduciendo el prototipado (mediante repeticiones). Además, en él se analizan otras opciones, y se realizan tareas para identificar y tratar de minimizar los riesgos.

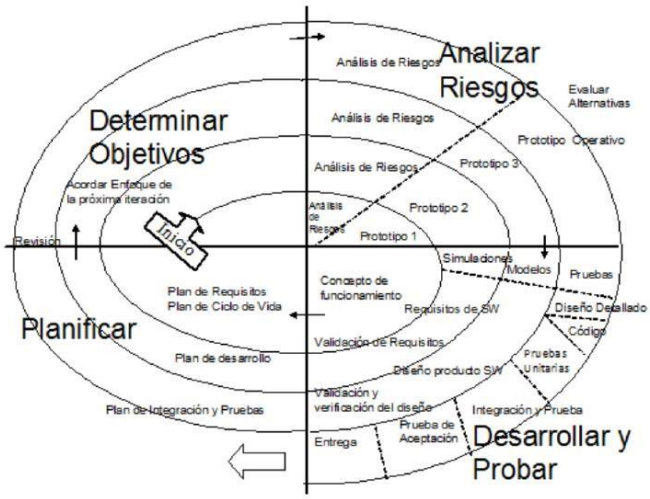

Figura 11. Modelo en Espiral (Rojas Contreras et al. 2011)

Tanto este modelo, como el iterativo, permiten una adaptabilidad mucho mayor que el modelo secuencial, pero no son pocas las personas, que creen que no son suficientes para la variabilidad del mercado: extensas planificaciones y periodos de análisis, juntamente con la excesiva documentación, llevan a que este tipo de modelos, se parezcan más a las metodologías tradicionales.

3.2 Contexto histórico.

Para explicar la aparición y evolución de las metodologías, debemos hacer referencia al contexto histórico relacionado con el desarrollo de la tecnología informática en últimos 60 años.

A finales de los años 60, los programadores encargados de codificar los ordenadores, y/o computadoras de la época, no prestaban gran atención al estilo de programación que utilizaban, debido a los limitados recursos de espacio y velocidad que prestaban dichas máquinas: su principal preocupación consistía en escribir un código de programación que ocupase el menor espacio posible y fuese eficiente. A todo esto, hay que sumar que los compiladores de la época eran bastante limitados en perjuicio de la calidad en la decodificación.

Esta situación, dio un vuelco cuando empezaron a aparecer los primeros equipos "pequeños" y "baratos", que venían como resultado de la **Ley de Moore**. Este hecho, cambió los criterios para medir la eficiencia de cualquier programa, ya que los equipos al ser más rápidos, y sin limitaciones de espacio, no requerían un código fuente tan eficiente y liviano como antes.

Con la aparación del *System/360* de IBM en 1964, -primeros pc´s para ejecutar aplicaciones, indistintamente de su tamaño o ambiente (científico o comercial)- y la estandarización de los lenguajes de alto nivel, se aseguró portabilidad y perdurabilidad sobre los programas, consiguiendo que los costes del software superasen a los costes del hardware, siendo el arranque de la **crisis del software** que hoy día perdura.

Figura 12. La IBM PC original (1981)

En base a estos hechos, se establecieron los siguientes criterios para marcar el éxito del desarrollo software:

1. El coste inicial debe ser relativamente bajo.
2. Debe ser fácil de mantener.
3. Debe de ser llevable a nuevo hardware.
4. Debe hacer lo que el cliente quiere.

A pesar de estas pautas, los programadores siguieron creando software sin hacer mucho caso. *Edsger Dijkstra* (1930-2002), - padre de la programación distribuida- propuso la computación como una parte de las matemáticas aplicadas y sentó las bases de la **programación estructurada**, origen de las metodologías para el desarrollo informático.

Dicha programación se basa en:

- Desarrollar utilizando la filosofía **top-down** (del proceso global, a los procesos concretos), en contraposición a **bottom-up** (funcionalidades detalladas se van componiendo y entrelazando).
- Uso de un conjunto de premisas para programar (utilizando lógicas iterativas tipo *Do* o *While* en lugar del *"Go to"*).
- Seguimiento de una serie de principios, o criterios, para descomponer problemas de mayor tamaño.

A raíz de este planteamiento de Dijkstra (a través de su libro en 1976, *A Discipline of Programming*), emergieron lenguajes de programación con claras referencias a sus bases, como el lenguaje **Pascal,** que trata de crear un código con una buena estructura y entendible. Pero Dijkstra no solo planteó la cuestión sobre la lógica basada en el comando "*Go to*", sino que, mediante sus principios, implantó la idea de diseñar de una forma estructurada, también llamada, Ingeniería del Software.

Durante la conferencia de la OTAN de 1968, los países que se reunieron para discutir los problemas de fracasos derivados del desarrollo informático usaron denominaciones como, "*ingeniería del software*", planteando soluciones, como la de formular unas pautas sustentadas en dos principios básicos en toda ingeniería:

1. Gestión predictiva: antes de comenzar un proyecto hay que clarificar qué se va a realizar: toma de requerimientos, planificación, y cálculo de costes y riesgos. Tras ello, realizar un control de riesgos para que estos se minimicen y el plan se lleve a cabo.

2. Principio de calidad: La calidad del producto, es el resultado de la suma de las calidades parciales de los pasos que llevan a la generación de dicho producto.

Todo esto, juntamente con una bajada del coste del hardware, hizo que se pusisera el foco en el coste del software: Los clientes comenzaron a molestarse por la cantidad de dinero invertido y los bajos resultados, que pedían inyectar más y más dinero. Fue el origen, de las metodologías de desarrollo del software.

3.3 Cronología de las metodologías.

El desarrollo ágil, y el conjunto de metodologías que lo componen, surge como un método necesario para afrontar la gestión de los proyectos, frente a los modelos clásicos de desarrollo en cascada, o *waterfall.*

La historia del nacimiento de las distintas metodologías, se puede dividir según Avison y Fitzgerald (Avison y Fitzgerald 1998) en:

1. **Pre-metodologías**: entre los años 1950-1960.
2. **Primeras metodologías**: entre los años 1970-1980.
3. **La era de las metodologías**: entre los años 80 y principios de los 90.
4. **Era post-metodológica**: a partir de los 90.

3.3.1 La era pre-metodológica

Está marcada por ser anterior a la aparición de las primeras metodologías, y en la que destaca la existencia de grandes computadoras, que se encargaban de ejecutar aplicaciones científicas y militares. De forma gradual, se fueron empleando en empresas, realizando pequeños informes económicos como resultado de una serie de cálculos, o para el almacenamiento de información de usuarios y/o documentos en ficheros.

Los sistemas que se empleaban fueron desarrollados por programadores, como resultado de ejercicios, para su ejecución en plazos cortos de tiempo. Los usuarios normalmente quedaban insatisfechos con las soluciones planteadas, al ser la documentación escasa o en ocasiones no existir. Además, cualquier modificación necesitaba tiempo haciendo más compleja la aplicación, transformando a los programadores en un bien cotizado, al ser los únicos que sabían el código.

Poco a poco, fue siendo necesario más tiempo para analizar y diseñar las aplicaciones, apareciendo las figuras de los analistas de sistemas, que hacían el papel de intermediario entre el programador y el sistema; y, los operadores de sistemas, dedicados a controlar su funcionamiento.

3.3.2 Las primeras metodologías

Comienzan en 1971 con la aparición del modelo en cascada (Royce 1970).Esta metodología, se basa en la realización de una serie de etapas, de forma que para considerar un producto como entregable, todas las etapas anteriores asociadas debían estar finalizadas. Royce presentó un modelo de siete etapas:

1. **Requerimientos del sistema.**
2. **Requerimientos de software.**
3. **Análisis.**
4. **Diseño del programa.**
5. **Codificación.**
6. **Pruebas.**

7. **Operaciones (Implantación y mantenimiento).**

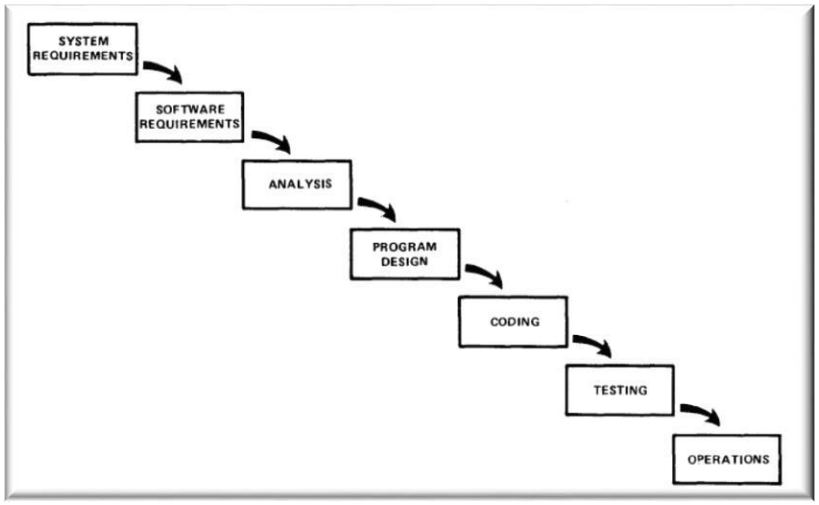

Figura 13. Esquema de etapas presentado por W. Royce

Las principales desventajas que plantean este modelo son el grado de satisfacción de los clientes con la entrega final (no suele ser acorde), la falta de flexibilidad a los cambios, el exceso de documentos, y la facilidad para no cumplir los tiempos estipulados.

3.3.3 La era de las metodologías

Nacieron nuevos enfoques en respuesta a las limitaciones enumeradas anteriormente, que se pueden agrupar en dos vertientes: las que mejoraron el modelo *waterfall* y las que buscaron una nueva forma de gestionar proyectos.
Además, a finales de los años 70 surgieron nuevas herramientas y técnicas sobre los que se basaron estas nuevas metodologías, como pudieron ser los diagramas de flujos, el modelo entidad-relación, la normalización, los diagramas de actividad, etc., y herramientas, como las de gestión de proyectos, bases de datos de código, repositorios de modelado, etc.
Entre las metodologías que mejoraron el modelo waterfall, surgieron MERISE, Yourdon System Method, o Structured Systems Analysis and Design Method (SSADM), que integraban las novedades enumeradas en el párrafo anterior.

Entre las metodologías alternativas de los años 80, Avison realizó una clasifición de las mismas en varios grupos de una forma sencilla de entender:

- **Sistemáticas:** emplean técnicas cualitativas a situaciones no sistemáticas. Un modelo que aplica esta medología podría ser Soft Systems Methodologies (SSM).
- **Estratégicas:** enfocadas en la planificación previa al desarrollo de software, para cumplir los requisitos de negocio. Un modelo que aplica esta medología podría ser Business Systems Planning.
- **Participativas:** basadas en la involucración de usuarios en el análisis, diseño e implementación. Como podría ser, ETHICS.
- **Prototipadas:** utilizan prototipos para visualizar y validar el producto. Un modelo que aplica esta medología podría ser, Rapid Application Development (RAD).
- **Estructuradas:** extienden la programación estructurada al análisis y diseño, con técnicas que permiten un análisis descendente y la representación compleja de procesos.
- **Análisis de datos:** tratan de documentar y comprender los datos. Un modelo que aplica esta medología podría ser, Information Engineering. Posteriormente, en los primeros años de los 90 nacieron nuevas metodologías (Carvajal Riola 2008) como:

- **Orientadas a objetos**. Enfocadas en objetos que representan entidades del mundo real.
- **Desarrollo incremental o evolutivo**. Basadas en prototipos primitivos sobre los que desarrollar por etapas. Un modelo que aplica esta medología podría ser, Dynamic System Development Methods (DSDM).
- **Específicas**. Llevadas a cabo para cierto tipo de proyectos, buscan un objetivo en concreto.

Sin embargo, gran parte de los usuarios de estas metodologías, no acabaron de estar satisfechos con las metodologías que emplearon, bien fuera *waterfall* o alguna de sus variantes.
La mayoría, estaban destinadas para aplicarse en situaciones ideales, y en las situaciones reales, aparecían problemas, debido también a la singularidad de cada proyecto.

Uno de los motivos principales de este descontento, fue que los usuarios de dichas metodologías esperaban encontrarse un guión paso a paso, y con un enfoque descendente (desde arriba hacia abajo) a lo largo del desarrollo (rara vez se da), pues lo normal, suele consistir en adaptar el guión, omitiendo fases y modificando el orden de ejecución de los pasos de éste.

El resultado fue la aplicación de metodologías más permisivas en las que el desarrollador elige la estructura, las herramientas, las técnicas, y las fases, en función de la necesidad del momento. (Un modelo que aplica esta metodología podría ser Multiview).

3.3.4 la era post-metodológica

El éxito o fracaso en el desarrollo no dependió únicamente a la utilización de una metodología en concreto, pero a pesar de ello, nació una corriente, que replanteó los beneficios de las metodologías, que, juntamente con otros conceptos no metodológicos, darían paso a esta etapa.

Existen diversos motivos que explicarían este rechazo al uso de las metodologías, pero una de las principales fue la decepción por el no cumplir las expectativas iniciales: las metodologías no pudieron solucionar, todos aquellos problemas para los que supuestamente la metodología pondría fin.

Algunas de las razones que Fitzgerald y Avison (Avison y Fitzgerald 1998) presentan como las causantes del debate metodológico fueron:

- **Productividad:** no aumentaron la productividad prometida.
- **Complejidad:** en ocasiones eran demasiado complejas.
- **Habilidades:** eran necesarias habilidades específicas.
- **Herramientas:** era difícil utilizarlas, caras y no eran rentables.
- **No contingente:** no se adaptaban a las necesidades.
- **Enfoque unidimensional:** basadas en una sola idea sobre cómo desarrollar.
- **Inflexible:** imposibilidad de adaptarse a los cambios.
- **Supuestos no válidos:** no se permite la modificación de requerimientos en ningún momento.
- **Desplazamiento del objetivo:** el uso literal de las herramientas, dejando de lado la creatividad.
- **Poco orientadas:** a personas y al entorno.
- **Dificultades:** para ponerlas en práctica.

Como se ha explicado hasta ahora, cuando una metodología, o varias, no cubrían las necesidades, surgían nuevas variantes que trataban de dar solución.

Mostramos ahora, los planteamientos que se llevaron a cabo en las empresas, y en los usuarios respecto a esta situación:

- ***Desarrollo externo:*** las empresas optaron por no realizar de forma interna proyectos informáticos y externalizarlos.
- ***Mejora continua*****:** se apuesta por la mejora continua.
- ***Desarrollo ad hoc[i] y contingencia***: el desarrollo *ad hoc* supone la vuelta a la era premetodológica.
- ***Desarrollo contingente o flexible*****:** permitiendo, manteniendo una cierta estructura, posibilidades de elección de componentes a los desarrolladores.
- ***Desarrollo ágil*****.** Centrada en la participación de los usuarios y clientes más que en procesos y herramientas, trabajando más en el software y menos en la documentación, colaborando más con clientes en vez de negociar, y responder a los cambios por encima de la planificación de los tiempos del trabajo.

3.4 Requisitos de una Metodología

La enciclopedia de la ingeniería de software (Scacchi 2002) propone los siguientes aspectos a cumplir por toda metodología:

1. **Visión del producto:** todas las partes deben de conocer todos los detalles sobre el producto: plazos, tareas, cambios, etc.
2. **Vinculación con el cliente:** la metodología trata de ser la base para indicar las relaciones entre los distintos actores dentro de un proyecto: usuarios, programadores, soporte, etc.
3. **Establecer un modelo de ciclo de vida:** elegir un modelo: iterativo, waterfall, etc. Así se podrán conocer las fases, y gestionar los recursos.
4. **Gestión de los requisitos:** mínima calidad exigible a los requerimientos del producto.
5. **Plan de desarrollo:** plan lo suficientemente claro, y conciso, para que los programadores puedan llevar a cabo los desarrollos de una forma clara y precisa, sin equivocaciones, y sin dudas. También debe ser posible añadir modificaciones con posibilidad de recuperar el plan original.
6. **Integración del proyecto:** La metodología debe clarificar como el producto se relacionará con el resto de los productos actuales, en el presente, y a futuro.
7. **Medidas de progreso del proyecto:** deben ser accesibles, y lo suficientemente claras, para que todas las partes del proyecto sean capaces de ver, de una forma fácil, y rápido el estado actual de un proyecto, y del tiempo que resta para su terminación.
8. **Métricas para evaluar la calidad:** el equipo encargado de gestionar los despliegues del producto debe controlar y preservar la calidad del software que se está desarrollando, como un símbolo de estabilidad del producto.
9. **Maneras de medir el riesgo**: se deben tener controlados los riesgos: no solo en la gestión de estos, en las fases donde aparezcan, sino también en la medida en que puedan ser cuantificados y cualificados para dar una mejor respuesta a los pasos necesarios a dar, para afrontarlos.

10. **Cómo gestionar los cambios:** debe ser posible incluir la posibilidad de que aparezcan cambios, o bien en las necesidades del negocio, o bien en el propio desarrollo del software.
11. **Establecer una línea de meta:** se debe tener presente con qué funcionalidades probadas, se dará por bueno un producto. Es necesario que todas las partes que conforman el proyecto sepan de forma clara, cuando algo está "finalizado".

Viendo todos estos aspectos enumerados por la enciclopedia del software, parece que no hay ninguno de ellos que se aleje del uso del sentido común, y con los que es fácil estar de acuerdo en que son necesarios en cualquier metodología.

3.5 Metodologías tradicionales

3.5.1 Introducción

En este capítulo se pretende introducir algunas de las metodologías de gestión de proyectos más representativas del grupo al que calificamos como metodologías tradicionales.
A lo largo del Máster se han estudiado algunas de ellas, como por ejemplo: PMBOK, ICB, PRINCE2.
Se ha hecho una introducción para cada una de ellas, hablando de la organización que la creó y su evolución histórica.
A continuación, se explica brevemente la estructura que sigue cada metodología.

Finalmente se reseñan muy brevemente otras 6 metodologías: ISO 21500, SWEBOK, SSADM, MERISE, MÉTRICA y APMBOK.

3.5.2 PMBOK

3.5.2.1 Introducción

La guía Project Management Body Of Knowledge (PMBOK) es el estándar de gestión de proyectos del Project Management Institute (PMI) y es el único estándar acreditado por la American National Standards Institute (ANSI) (organismo para la coordinación y el uso de estándares de EEUU). Proporciona un marco común para los gestores de proyectos, suministrando un léxico y unos procedimientos estructurados aplicables a cualquier tipo de proyecto.

PMI se fundó en 1960 y es una organización internacional sin ánimo de lucro que asocia a profesionales relacionados con la gestión de proyectos. A finales de 1970 casi 2000 miembros formaban parte de la organización y en los 80 se realizó la primera evaluación para la certificación como profesional en gestión de proyectos, llamado Project Management Professional (PMP). Desde 2011 es la más grande del mundo por estar integrada por más de 700.000 miembros de cerca de 170 países.

En este momento, el Project Management Institute (PMI®) está inmerso en una fase de cambios y actualizaciones (cambio de imagen corporativa, cambio de Partner de los centros de examen, cambio de formato del examen de certificación PMP a partir del 1 Enero 2021, etc) han anunciado que también van a cambiar la versión de PMBOK®.En 1996 se publicó la primera edición de PMBOK y actualmente está en su sexta edición publicada en 2016.
Normalmente, cada 4 años aproximadamente aparece una versión actualizada del Project Management Body Of Knowledge.

Parece ser que esta nueva versión del PMBOK® va a ser muy diferente con respecto a las versiones anteriores. El nuevo PMBOK® será un estándar basado en principios en lugar de en procesos (entradas, herramientas y salidas).

El énfasis a partir de ahora estará en los resultados y no tanto en los entregables. La innovación tecnológica que se ha producido en los últimos 10 años ha traído nuevos modelos de negocio, nuevas formas de trabajar, nuevas estructuras de equipo, y la necesidad de un abanico más amplio de gestión de proyectos (predictiva, ágil, híbrida) y desarrollo de nuevos productos. Es por ello, que el foco se ha movido a la aportación de resultados más que a las entregas.
El nuevo estándar enfatiza que los proyectos no deben, simplemente, producir entregables, sino que deben permitir que dichos entregables produzcan resultados que en último lugar reporten valor a las organizaciones y a los interesados.

Otra novedad que se suma a las anteriores, es que el PMBOK 7 será más ligero en contenido, dando paso a la plataforma interactiva digital Standards Plus™ donde el PMI® compartirá contenidos online acerca de prácticas, métodos, artefactos e información de interés de manera dinámica.
El PMBOK® deja de ser por tanto un gran libro estático que cambia cada dos años (aproximadamente), a ser un contenido vivo y actualizado de manera continua.

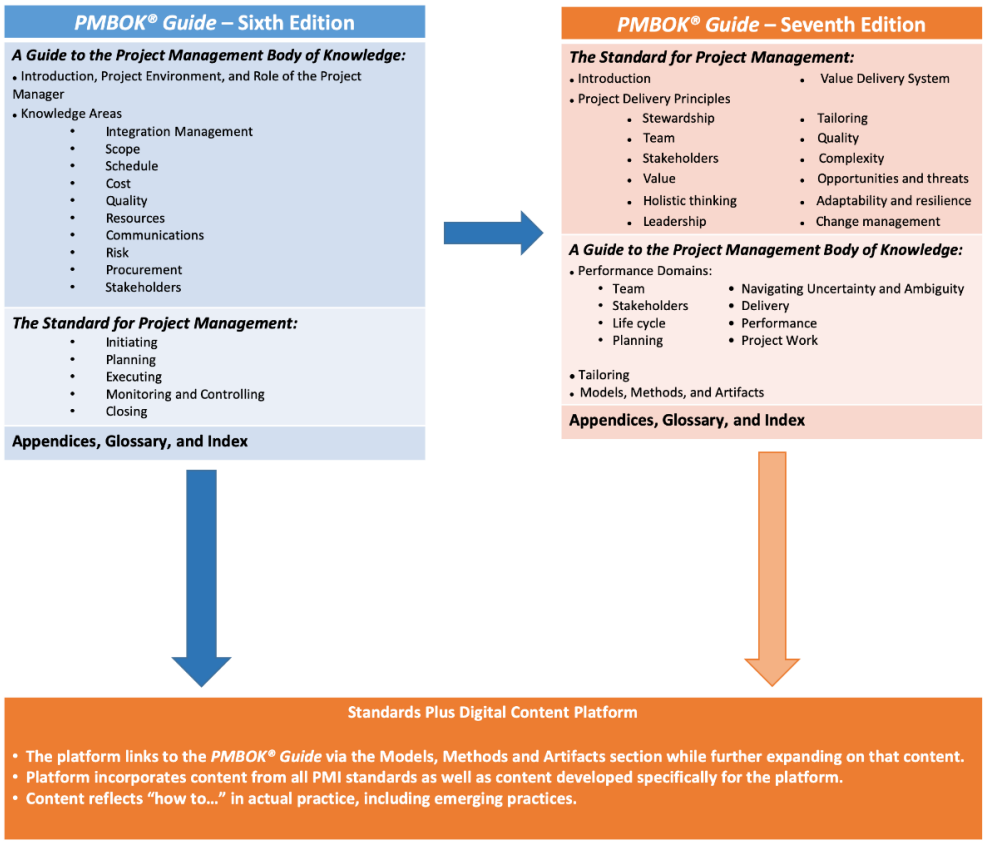

Figura 14. Cambios entre el PMBOK® Guide – Sixth Edition y el PMBOK® Guide – Seventh Edition

En la actualidad PMI tiene **ocho certificaciones**:

1. Project Management Professional
2. Certified Associate in Project Management
3. Program Management Professional

4. Portfolio Management Professional
5. PMI Agile Certified Practitioner
6. PMI Professional in Business Analysis
7. PMI Risk Management Professional
8. PMI Scheduling Professional

3.5.2.2 Estructura

Como hemos dicho, de momento PMBOK es una metodología basada en procesos. Esto es, describir el trabajo por paquetes que se llevan a cabo en procesos.
Este enfoque es similar a otras normas de gestión, tales como ISO 21500, ISO 9000 o Capability Maturity Model Integration (CMMI) del Software Engineering Institute (SEI). Los procesos se superponen e interactúan a lo largo de un proyecto en sus diversas fases. Los procesos se describen en términos de:

- Entradas: documentos, planos, diseños, etc...
- Herramientas y técnicas: son las transformaciones aplicadas a las entradas que producirán las salidas.
- Salidas (idem a las entradas): documentos, planos, diseños, etc...

La guía establece 49 procesos que se dividen en 5 grupos de procesos básicos y 10 áreas de conocimiento (típicas en la mayoría de los proyectos). Las definiciones de los 5 grupos de procesos se explican a continuación y sus interacciones se muestran en la figura 3.1

1. **Inicio**. Aquellos procesos realizados para definir un nuevo proyecto o una nueva fase de un proyecto existente.

2. **Planificación**. Aquellos procesos requeridos para establecer el alcance del proyecto, refinar los objetivos y definir la estrategia para alcanzar los objetivos que se marcaron al comienzo del proyecto.

3. **Ejecución**. Aquellos procesos realizados para completar el trabajo definido en el plan del proyecto y así satisfacer sus especificaciones.

4. **Monitorización** y **Control**. Aquellos procesos requeridos para realizar el seguimiento, revisar y controlar el progreso y la eficiencia del proyecto. Se identifican las áreas en las que los cambios en el plan son necesarios y se inician los cambios correspondientes.

5. **Cierre**. Aquellos procesos realizados para finalizar formalmente todas las actividades de todos los procesos.

Las 10 áreas de conocimiento son las siguientes

- **Gestión de la Integración**. Se incluyen todas las actividades y procesos que hay que realizar para identificar, combinar y coordinar los diversos procesos y actividades de gestión dentro de los grupos de gestión de procesos.
- **Gestión del Alcance**. Se incluyen los procesos para asegurar que el proyecto tiene todo el trabajo necesario y sólo el necesario, para completar el proyecto de forma satisfactoria.

- **Gestión de Plazos**. Se incluyen los procesos requeridos para finalizar el proyecto de forma satisfactoria en el plazo previsto.
- **Gestión de Costes**. Se incluyen los procesos necesarios para poder planificar, estimar, presupuestar y controlar los costes de forma que se pueda finalizar dentro de los costes planificados.
- **Gestión de la Calidad**. Se determinan las políticas de calidad, objetivos y responsabilidades de forma que el proyecto satisfaga las necesidades previstas.
- **Gestión de los recursos**. Se encarga de organizar y gestionar al equipo de proyecto, asignando los roles y responsabilidades correspondientes.
- **Gestión de la Comunicación**. Se asegura la generación temporal apropiada y la distribución, colección y almacenamiento de la información del proyecto.
- **Gestión del Riesgo**. Son los procesos que realizan la planificación, identificación, análisis cualitativo y cuantitativo de los riesgos, así como la planificación de las medidas a adoptar y su control.
- **Gestión de Aprovisionamiento**. Son los procesos que incluyen la adquisición de productos, servicios o resultados necesarios y que siendo ajenos al equipo del proyecto son necesarios para el trabajo a realizar.
- **Gestión de Involucrados**. Se incluye los procesos requeridos para identificar a todas las personas u organizaciones afectadas por el proyecto, analizando las expectativas y el impacto de las partes interesadas en el proyecto y el desarrollo de estrategias de gestión adecuadas para la participación efectiva de los interesados en las decisiones y la ejecución de proyectos.

Procesos PMBOK 6	Grupo de Procesos de Iniciación	Grupo de Procesos de Planificación	Grupo de Procesos de Ejecución	Grupo de Procesos de Monitoreo y Control	Grupo de Procesos de Cierre
Gestión de la Integración del Proyecto	Desarrollar el acta de constitución del proyecto	Desarrollar el Plan para la Dirección del Proyecto	Dirigir y Gestionar el Trabajo del Proyecto Gestionar el Conocimiento del Proyecto	Monitorear y Controlar el Trabajo del Proyecto Realizar el control Integrado de Cambios	Cerrar Proyecto o Fase
Gestión del Alcance del Proyecto		Planificar la Gestión del Alcance Recopilar Requisitos Definicir el Alcance Crear la EDT - Estructura de Desglose de Trabajo WBS		Verificar el Alcance Controlar el Alcance	
Gestión del Cronograma del Proyecto		Planificar la Gestión del Cronograma Definir las Actividades Secuenciar las Actividades Estimar de la Duración de las Actividades Desarrollar el Cronograma		Controlar el Cronograma	
Gestión de los Costes del Proyecto		Planificar la Gestión de Costos Estimar los Costos Determinar el Presupuesto		Controlar los Costos	
Gestión de la Calidad del Proyecto		Planificar la Gestión de Calidad	Gestionar la calidad	Controlar la Calidad	
Gestión de los Recursos del Proyecto		Planificar la Gestión de Recursos Estimar los recursos de las actividades	Adquirir Recursos Desarrollar el Equipo Dirigir el Equipo	Controlar los Recursos	
Gestión de las Comunicaciones del Proyecto		Planificar la Gestión de las Comunicaciones	Gestionar las Comunicaciones	Monitorizar las Comunicaciones	
Gestión de los Riesgos del Proyecto		Planificar de la Gestión de Riesgos Identificar los Riesgos Realizar el Análisis Cualitativo de los Riesgos Realizar el Análisis Cuantitativo de los Riesgos Planificar la Respuesta a los Riesgos	Implementar la respuesta a los riesgos	Monitorizar los Riesgos	
Gestión de las Adquisiciones del Proyecto		Planificar la Gestión de Adquisiciones del Proyecto	Efectuar las Adquisiciones	Controlar las Adquisiciones	
Gestión de los Interesados del Proyecto	Identificar a los Interesados	Planificar la Pacticipación de los Interesados	Gestionar Participación de los Interesados	Monitorizar la Participación de los Interesados	

Figura 15. Procesos de PMBOK 6

3.5.3 IPMA Competence Baseline (ICB)

3.5.3.1 Introducción

IPMA Competence Baseline (ICB) es el estándar de International Project Management Association (IPMA) para la competencia en la dirección de proyectos.

IPMA se creó en 1965 en Suiza siendo la organización más antigua de gestión de proyectos. Está formada por una red de asociaciones nacionales. Es decir, se constituye como la organización representativa de todas las asociaciones nacionales instaladas en cada país, que orientan sus servicios a las necesidades nacionales de desarrollo en el área de gestión de proyectos y en su misma lengua. La asociación española se llama Asociación Española de Ingeniería de Proyectos (AEIPRO) que es otra organización sin ánimo de lucro creada en 1992. También tiene suscrito un convenio de colaboración con el PMI.

IPMA comenzó con una versión inicial de ICB en 1995, definiendo y validando la competencia de los directores de proyectos. En 1999 se publicó la versión ICB 2.0 y en 2006 se renovó con ICB 3.0 [3] que sigue vigente en la actualidad.

IPMA tiene para individuos 4 niveles de **certificación**, de menos a más avanzado:

1 Técnico en dirección de proyectos (IPMA nivel D),
2 Profesional en dirección de proyectos (IPMA nivel C),
3 Director de proyecto (IPMA nivel B) y
4 Director de cartera de proyectos (IPMA nivel A).

3.5.3.2 Estructura

ICB contiene los términos básicos, tareas, habilidades, funciones, procesos, métodos, técnicas y herramientas que se deben usar, tanto teórica como prácticamente, para una buena gestión de proyectos. El objetivo principal de ICB es estandarizar y reducir las tareas fundamentales necesarias para completar un proyecto de la forma más efectiva y eficiente.

La estructura de ICB es bastante distinta a las demás metodologías porque se organiza por competencias, no por procesos. Los 46 elementos de competencia se agrupan en 3 grandes grupos (ver figura 16).

- Competencias técnicas. Abarcan el cumplimiento de los requisitos del proyecto (o programa o portfolio) según los involucrados, la integración de los trabajos de la organización (sea en un proyecto temporal, programa o portfolio) y la producción de entregables simples en la organización del proyecto. Las competencias son 20:

  Éxito en la dirección de proyectos; Partes involucradas; Requisitos y objetivos de proyectos; Riesgos y oportunidades; Calidad; Organizaciones de proyectos; Trabajo en equipo; Resolución de problemas; Estructuras de proyectos; Alcance y entregables; Tiempo y fases de los proyectos; Recursos; Coste y financiación; Aprovisionamiento y contratos; Cambios; Controles e informes; Información y documentación; Comunicación; Arranque; Cierre.

- Competencias de comportamiento. Abarcan las competencias relacionadas directamente con el propio director del proyecto, las derivadas de la gestión del director, las comunes y contextuales utilizados en la gestión global del proyecto y sus involucrados y las competencias que tienen sus orígenes en la economía, la sociedad, la cultura y la historia. Las competencias son 15:
- Implicación; Autocontrol; Asertividad; Relajación; Accesibilidad; Creatividad; Liderazgo; Orientación al resultado; Eficiencia; Consultable; Negociación; Crisis y conflictos; Credibilidad; Apreciación de valores; Ética.
- Competencias contextuales. Se agrupan en términos del rol de la gestión de proyectos en las organizaciones permanentes y de las interrelaciones de gestión de proyectos con la administración de negocios. Las competencias son 11:

  Orientación al proyecto; Orientación al programa; Orientación al portfolio; Implementación de proyecto, programa y porftolio; Legalidad; Negocio; Organización permanente; Sistemas, productos y tecnología; Gestión de personal; Salud, seguridad, prevención y entorno; Financiación.

Figura 16. Competencias para la gestión de proyectos de ICB

3.5.4 PRINCE2

3.5.4.1 Introducción

La metodología Projects in a Controlled Environment (PRINCE2) deriva del método de gestión de proyectos PRINCE, que fue desarrollado inicialmente en 1989 por la Central Computer and Telecommunications Agency (CCTA) como un estándar del gobierno del Reino Unido para los sistemas de IT de gestión de proyectos. A su vez, PRINCE proviene de la metodología Project Resource Organisation Management Planning Technique (PROMPT) creada a mediados de los 70 por la empresa Simpact Systems Limited que proporciona un marco adecuado para gestionar la estrategia, viabilidad, desarrollo y apoyo de los sistemas de IT a través de un enfoque estructurado de gestión de proyectos. Por tanto, esta metodología tiene una especial significación en este TFM por sus orígenes ligados al sector Tecnologías de la Información y Comunicación (TIC).

PRINCE es un acrónimo de “proyectos en ambientes controlados” pero pronto llegó a ser aplicado regularmente fuera del entorno TIC, tanto en el gobierno del Reino Unido como en el sector privado. PRINCE2 fue lanzado en 1996 como un método genérico de gestión de proyecto alcanzando cotas de bastante popularidad por ser un estándar de facto para la gestión de proyectos en varios países (sobre todo de la Commonwealth), empresas multinacionales y organizaciones internacionales.

En 2009 se publicó una revisión "PRINCE2: 2009 Refresh" manteniendo el nombre (en lugar de PRINCE3 o similar) para indicar que el método sigue siendo fiel a sus principios. Sin embargo, se trata de una revisión importante de la versión de 1996 para adaptarla al entorno empresarial cambiante transformándolo en una metodología más simple y ligera. PRINCE2 esta perfectamente alineada con la norma ISO 21500.

En 2013 la propiedad de los derechos de PRINCE2 se transfirió del organismo público Británico HM Cabinet Office a Axelos Ltd, una empresa conjunta de HM Cabinet Office y la empresa privada Capita plc. Hay 3 niveles de certificación, de menos a más avanzada:
Foundation Examination, Practitioner Examination y Professional Examination.

3.5.4.2 Estructura

La metodología PRINCE2 se apoya en 7 principios, enriqueciendo no sólo al proyecto en concreto, sino a toda la organización en la que se desarrolla.

- Justificación comercial continua. Se asegura que hay un motivo justificable para iniciar el proyecto y que perdura durante toda su vida.
- Aprender de la experiencia. Se recogen experiencias anteriores, las que se van obteniendo durante el proyecto y las lecciones aprendidas a su cierre.
- Roles y Responsabilidades definidos. Asegurando que los involucrados del proyecto están representados en la toma de decisiones.
- Gestión por Fases. El proyecto se planifica, supervisa y controla fase a fase.
- Gestión por excepción. Delegar la autoridad de gestión al subordinado, dándole autonomía según unas tolerancias pautadas (de plazos, coste, calidad, alcance, beneficio y/o riesgo) de manera que si se sobrepasa la tolerancia, se consulte al superior como actuar.
- Orientación a productos. Concentrarse en la definición y entrega de productos. Es decir, un proyecto no son un conjunto de tareas, sino que se entregan productos (que se elaboran tras la ejecución de las tareas necesarias).
- Adaptación. Se asegura que la metodología y los controles a aplicar se basan en el tamaño, complejidad, importancia, capacidad y nivel de riesgo del proyecto.

Hay 8 grupos de procesos que se resumen a continuación y sus interacciones se esquematizan en la figura 17.

1. Emprender un proyecto (SU). Se realiza al comienzo del proyecto siendo una preparación inicial para la gestión del proyecto, su control y viabilidad. Este proceso lo crea la Junta del proyecto garantizando los recursos necesarios.

2. Dirigir Proyecto (DP). Dirige y define las responsabilidades de la Junta en la supervisión del proyecto. Está por encima de todos los procesos, interactuando con el resto. Proporciona los protocolos de aprobación al final de cada etapa y al cierre del proyecto. Este proceso es el marco de suministro de entradas, de recepción de requisitos y toma de decisiones. Es el único proceso en el que actúa la Junta porque el resto de procesos son llevados por el Director del proyecto y su equipo.

3. Iniciar proyecto (IP). Sólo se realiza una vez durante el ciclo de vida del proyecto. Sirve para plantear cómo se puede gestionar la totalidad del proyecto y se plasma en el documento de inicio del proyecto (Project Initiation Document, PID). El objetivo del documento es el establecimiento de un entendimiento común de los elementos críticos del proyecto, así como el acuerdo de la Junta para la primera etapa de desarrollo del proyecto.

4. Planificación (PL). Proceso común para el resto de procesos. Los planes se producen identificando los entregables del proyecto, las actividades y recursos necesarios para crearlos. Todo ello tiene en una relación consistente con los requerimientos identificados en el PID.

5. Control de etapa (CS). Provee una guía para la gestión diaria del proyecto. Incluye la autorización y recepción de trabajos, gestión del cambio y de versiones, análisis e informes, consideraciones de viabilidad, acciones correctivas y escalado de incidencias a la Junta. Este proceso de control se realiza de forma iterativa por cada etapa de desarrollo.

6. Gestión de entrega del producto (MP). Forma parte del sistema de autorización siendo el mecanismo que sirve para que los ejecutores del trabajo técnico acuerden los trabajos a realizar. Se repite por cada paquete de trabajo autorizado.

7. Gestión de los límites de etapa (SB). Realizar la transición de un estado finalizado al inicio del siguiente estado, garantizando que el trabajo finalizado se ha realizado de acuerdo con los requisitos establecidos.

8. Cerrar un proyecto (CP). Proceso de transición de entrega del proyecto a la organización. Puede finalizar por haber realizado el trabajo satisfactoriamente o por terminación prematura, guardando las lecciones aprendidas. El proceso permite garantizar que si el cierre es por finalización del trabajo, ésta satisface las necesidades del cliente.

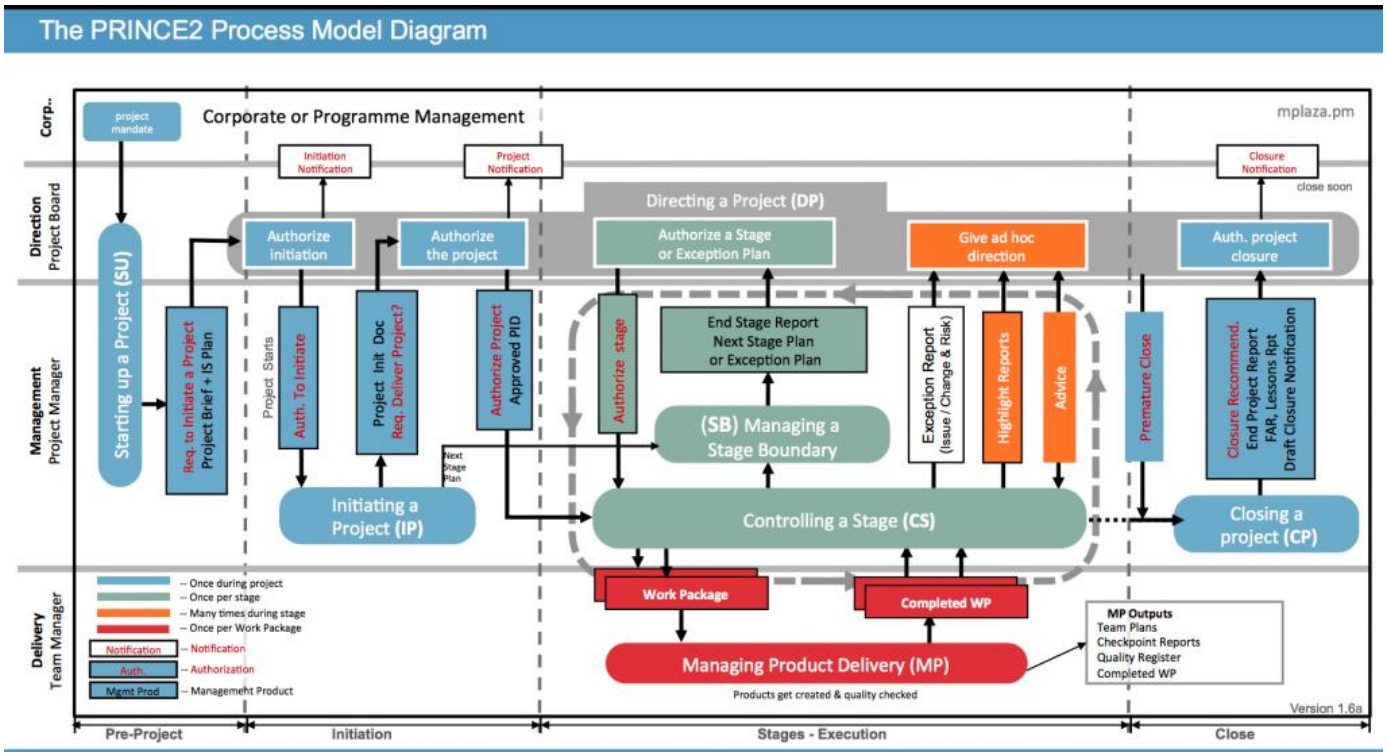

Figura 17. PRINCE2 Process Model Diagram

Por último, hay 8 áreas de gestión de proyectos que deben abordarse de forma continua durante todo el proyecto.

- Caso de negocio. Desarrollar una idea en una propuesta viable para la organización. La gestión del proyecto mantiene la atención en los objetivos de la organización y en los beneficios durante todo el proyecto. Es decir, responde a ¿por qué?

- Organización. Describe las funciones y responsabilidades del equipo (temporal) necesario para gestionar el proyecto con eficacia. Es decir, responde a ¿quién?

- Calidad. Los requerimientos de calidad de los entregables se basan en las descripciones del producto que a su vez son preparados por el Director del proyecto y aprobados por la Junta. Es decir, responde a ¿qué?
- Planes. Describen los pasos necesarios para entregar los productos, las técnicas que se deben aplicar y la comunicación a lo largo del proyecto. Es decir, responde a ¿cómo, cuánto y cuándo?
- Controles. El objetivo es garantizar que el proyecto genera los productos necesarios definidos en los criterios de aceptación y que cumple la planificación con los recursos y costes estimados. Además, debe garantizar la viabilidad del proyecto. Es decir, responde a ¿se ajusta a lo planificado?
- Riesgos. Aborda la forma de gestionar las incertidumbres en los planes y, en general, en el entorno del proyecto. Es decir, responde a ¿qué sucede si...?
- Cambio. El control de los cambios del alcance calcula el impacto de los potenciales cambios, su importancia, costes, impacto en el negocio y decidir si se incluyen o no. Es decir, responde a ¿cuál es el impacto?
- Progreso. La principal condición de control de un proyecto PRINCE2 es la existencia de un caso de negocio viable. Explica el proceso de toma de decisiones para la aprobación de planes y supervisar el rendimiento determinando si el proyecto debe continuar y cómo. Es decir, responde a ¿dónde estamos ahora?, ¿a dónde vamos? y ¿debemos continuar?

3.5.5 ISO 21500

La ISO 21500 es un estándar internacional desarrollado por la International Organization for Standardization (ISO) a partir de 2007 y publicado en 2012.
Proporciona una orientación genérica, explica los principios básicos y lo que constituye unas buenas prácticas en la gestión de proyectos.
Es un documento de orientación que no está destinado a ser utilizado con fines de certificación.
ISO tiene previsto que esta norma sea la primero de una familia de normas de gestión de proyectos y la diseño también para alinearse con otros estándares relacionados, tales como la ISO 10006:2003, ISO 10007:2003 e ISO 31000:2009

Podemos decir que la guía PMBOK y la norma ISO 21500 tienen grandes similitudes, por ejemplo, el conjunto de 10 áreas de conocimiento de PMBOK tiene su correspondencia exacta con la norma ISO 21500.

Sin embargo, como hemos visto a lo largo del máster, hay una serie de diferencias que podemos resumir a continuación:

- Hay 39 procesos en la ISO 21500 y 47 procesos PMBOK.
- 33 procesos de ISO 21500 tienen sus equivalentes directos en PMBOK.
- 4 procesos se han movido entre áreas de conocimiento.
- pares de procesos de PMBOK se han fusionado en 4 procesos individuales de la ISO 21500.
- 8 procesos del PMBOK están ausentes en la ISO 21500: 5.1 Plan de gestión del alcance, 5.5 Validar alcance, 6.1 Plan de gestión del calendario, 7.1 Plan de gestión de costes, 9.1 Plan de gestión de RRHH, 12.1 Plan de gestión de riesgos, 13.2 Plan de gestión de los involucrados y 13.4 Control del compromiso de los involucrados.

- 3 nuevos procesos se han introducido en la ISO 21500: 4.3.8 Recoger las lecciones aprendidas, 4.3.17 Definir la organización del proyecto y 4.3.19 Control de recursos.

3.5.6 SWEBOK

En el ámbito de los proyectos IT existen también algunas metodologías específicas como por ejemplo, El Software Engineering Body of Knowledge (SWEBOK) que ha sido promovido y realizado por el Institute of Electrical and Electronics Engineers (IEEE) Computer Society siendo publicado su última versión (V3) en 2014.
Corresponde con un estándar internacional, ISO/IEC TR 19759:2005.

En la edición de 2014, se definen 15 áreas de conocimiento:

- Requisitos de Software
- Diseño de Software
- Construcción de Software
- Pruebas de Software
- Mantenimiento de Software
- Gestión de la configuración
- Gestión de la Ingeniería de Software
- Proceso de Ingeniería de Software
- Herramientas y métodos de la Ingeniería de Software
- Calidad del Software
- Práctica Profesional de la Ingeniería de Software
- Economía de la Ingeniería de Software
- Fundamentos de Computación
- Fundamentos Matemáticos
- Fundamentos de Ingeniería

3.6 Metodologías Ágiles.

3.6.1 Necesidad de las metodologías Ágiles.

En la industria del software, el desarrollo de procesos, y por tanto de productos, ha venido utilizando metodologías que lo encasillaban como algo rígido e inflexible, aspecto que se desmiente al analizar la variedad y la dinámica del mercado de software. Como indica Boehm (Boehm 1979) en su libro *"Software Engineering as It Is"* la tendencia del sector se dirige hacia desarrollo en menos tiempo, y a una vida más corta de los productos finales. Es en estas condiciones, donde la entrega de valor de producto en plazos de tiempos muy cortos para llegar al mercado, marcan la diferencia.

Sin embargo, las metodologías tradicionales, presentan ciertos problemas a la hora de gestionar proyectos cambiantes, o con cierto dinamismo. Algunos de estos problemas son:

- La toma de requerimientos como primera etapa ante de comenzar el desarrollo, debe cubrir todas las funcionalidades y todos los aspectos de aquello que se desea implantar. Con esto, se busca evitar cambios a mitad de proyectos, que resulten inviables por costes y tiempo (también se conocen como metodologías predictivas). La realidad de este hecho es que difícilmente una toma de requerimientos completa justifica el tiempo y coste invertido en realizarla.

- El cliente no tiene los conocimientos suficientes para definir con el nivel de detalle necesario, lo que necesita, cambiando sus necesidades a lo largo del desarrollo. Llevar un control de cambios, y distintas herramientas para controlar este tipo de situaciones, permiten proteger al proyecto en su desarrollo, pero desde el punto de vista del cliente, se entiende la gestión como algo que no se adapta a sus necesidades cambiantes, y que, en caso de necesidad de cambios, tengan costes adicionales.

- El desarrollo del proyecto contiene documentación que no siempre se utiliza, ni tampoco se revisa. La entrega por hitos suele implicar la dedicación de recursos a elaborar mucha documentación con gran nivel de detalle que, en ocasiones, ni se utiliza, no la revisa el usuario, y por tanto no aporta valor sobre el producto final, teniendo que hacerse un esfuerzo exponencial en caso de que surjan cambios sobre el proyecto.

- El ritmo para desarrollar un software es lento. A los programadores, les resulta difícil entender sistemas tan complejos e interconectados, lo que desencadena en ritmos más lentos. Por ende, los propios desarrollos se abordan por etapas en pequeños trozos para reducir la posibilidad de errores y de costes a la hora de desarrollar. Además, las funcionalidades se van agregando, de forma que se puedan ir liberando parcialmente en los distintos entornos para que puedan ser probadas. Sin embargo, el ritmo que marca competencia exige velocidad, calidad y ahorro de costes, desplegando de manera continua funcionalidades, debido a que, si no, los productos en su totalidad puedan quedar obsoletos. Todos estos factores, hacen que sea difícil cumplir el ***Time to Market*** consiguiendo un malestar por parte de los usuarios:

- Al no haber sabido responder a tiempo, a lo que demandaba el mercado.

- A enfrentarse con los gestores de proyecto por no haberse involucrado lo suficiente para entregar el software a tiempo.

- A tener una sensación de que los desarrollos siempre son muy lentos, y muy costosos, pensando que faltan más programadores para adelantar dichos procesos de desarrollo.

3.6.2 Manifiesto Ágil: Origen, y principios.

El origen de la metodología ágil se remonta a febrero de 2001, donde durante una reunión celebrada en Utah (Estados Unidos), nació el uso de la palabra "ágil". En dicho evento, participaron 17 expertos de la industria del software, incluyendo a fundadores de algunas metodologías. El propósito de la reunión fue determinar los pasos a seguir por los grupos de desarrolladores para dar respuesta a la necesidad de crear un producto en el menor tiempo posible, admitiendo modificaciones durante su desarrollo. De esta forma, se buscaba una nueva forma de desarrollar software frente a los modelos tradicionales, más rígidos e inflexibles.

Como resultado, se fundó ***"La Alianza Ágil"***, una organización dedicada a promulgar estos principios, ayudando a las empresas a aplicar estos conceptos. La base sobre la que se construyó dicha alianza fue el Manifiesto Ágil, un documento que resume la filosofía "ágil" en 12 principios: los dos primeros sintetizan la idea de agilidad, y el resto, desarrollan las pautas de desarrollo para el equipo de programadores, su organización y cómo conseguir los objetivos (José, H., Canós, P. L., & Carmen 2003):

De forma análoga, a lo ya comentado anteriormente en los requisitos para el nacimiento de una nueva metodología, en estos principios lo que se pretende es aplicar de nuevo un poco de sentido común, planteando cuestiones elementales, basadas en experiencias previas de fracasos que generalmente coinciden siempre en los mismos aspectos.

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporten valor.

2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.

5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan, y confiar en ellos para conseguir finalizar el trabajo.

6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

7. El software que funciona es la medida principal de progreso.

8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

9. La atención continua a la calidad técnica y al buen diseño, mejora la agilidad.

10. La simplicidad es esencial.

11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

Estos 12 principios, se resumen en 4 aspectos que valoran las metodologías ágiles:

1. Se valora al **individuo y las interacciones del equipo** de desarrollo **sobre el proceso y las herramientas**. El éxito depende de las personas: es mejor tener un gran equipo, que un buen entorno de trabajo. En ocasiones se comienza preparando un entorno de desarrollo sobre el que se selecciona al equipo, esperando que éstos se adecúen a él, en lugar de dejar que sea el propio equipo el que defina sus condiciones de desarrollo.
2. **Desarrollar software** en lugar de tener una detallada documentación. Se aboga por "no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante". Los documentos han de ser de poca extensión y muy concretos.
3. **La colaboración con el cliente más que la negociación de un contrato**. Se plantea la relación continua entre usuario/negocio y equipo de programación. Esto hará que el proyecto fluya hacia un entendimiento total, y por tanto al éxito.
4. **Responder a los cambios más que seguir estrictamente un plan**. Es necesario poder integrar las modificaciones que surjan a lo largo del desarrollo (ya sea al principio, o durante el proyecto) para poder llevar a cabo con éxito el proyecto, siendo necesaria una gran flexibilidad en la planificación.

3.6.3 Tipos de Metodologías Ágiles.

En este apartado, se ha querido presentar las distintas metodologías Ágiles, para posteriormente, analizando los marcos de trabajo donde se mueven estas metodologías, realizar un análisis profundo de **Scrum**, que será la que se desarrolle con este proyecto.

3.6.3.1 Agile Project Management (APM)

Fue desarrollada por Jim Highsmith en 1999 a través de su libro *"Adaptive Software Development: A Collaborative Approach to Managing Complex System"*. En el libro, Jim defiende que el proceso de desarrollo debe estar en consonancia con los objetivos, de forma que, si los objetivos se repiten y se pueden definir de forma previa en su totalidad, un proceso prescriptivo sería idóneo, pero si los objetivos de negocio son cambiantes, y con cierto carácter innovador, entonces el marco de trabajo debiera ser ágil, con una gran flexibilidad y adaptabilidad.

El modelo APM se basa en 5 fases: la previsión, la especulación, la exploración, la adaptación y el cierre.

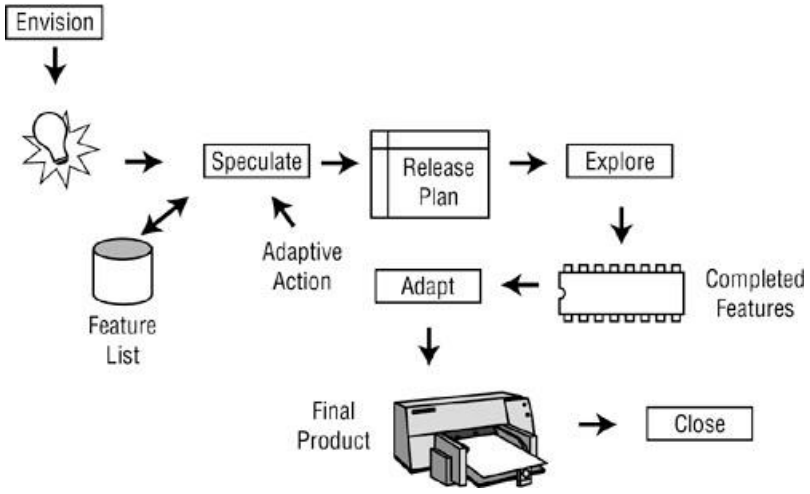

Figura 18. Fases del modelo APM. (Carvajal Riola 2008)

Previsión.

En esta fase, se definen los objetivos del proyecto, componentes del equipo y las formas de trabajar de forma conjunta.

Especulación.

En esta fase, se hace una planificación de una serie de entregables con el conjunto de funcionalidades del producto (esta fase reemplaza a la planificación tradicional, porque planificar implica predecir en cierta forma el futuro, mientras que la especulación permite designar el futuro como algo incierto).

Exploración.

En esta fase, se obtienen las funcionalidades requeridas, para ser probadas y aceptadas por el usuario final. En esa fase se da mayor importancia a la integración del equipo para el desarrollo de las funcionalidades.

Adaptación.

En esta fase, se revisan las funcionalidades desplegadas, el estado y cómo ha trabajo el equipo, haciendo algunas modificaciones si fuera necesario.

Cierre.

Terminado el proyecto, se celebra aprendiendo de las experiencias vividas.

3.6.3.2 Extreme Programming (XP).

La programación extrema fue desarrollada por Kent Beck en el año 2000, y propuesta en el libro *"Extreme Programming Explained: Embrace Change"* (Beck 2000).
Esta metodología, se centra en desarrollar las relaciones entre personas como forma de llegar al éxito en cualquier desarrollo. Por ello esta metodología tiene un diseño muy acorde a lo que se demanda dentro del desarrollo ágil.
En el Extreme Programming se busca la cooperación entre los distintos miembros del equipo, rodeados de un gran ambiente de trabajo.
XP se basa en el feedback continuo entre usuarios y programadores, dando paso a una comunicación fluida entre los miembros, busca la simplicidad de las soluciones y la determinación ante los cambios.
Su nombre tiene relación, a los principios que defiende, aplicando el sentido común, pero de una forma extrema.
Esta metodología es muy útil en proyectos donde los requisitos puedan ser imprecisos, variables, y donde los riesgos sean altos (Ortiz 2010)

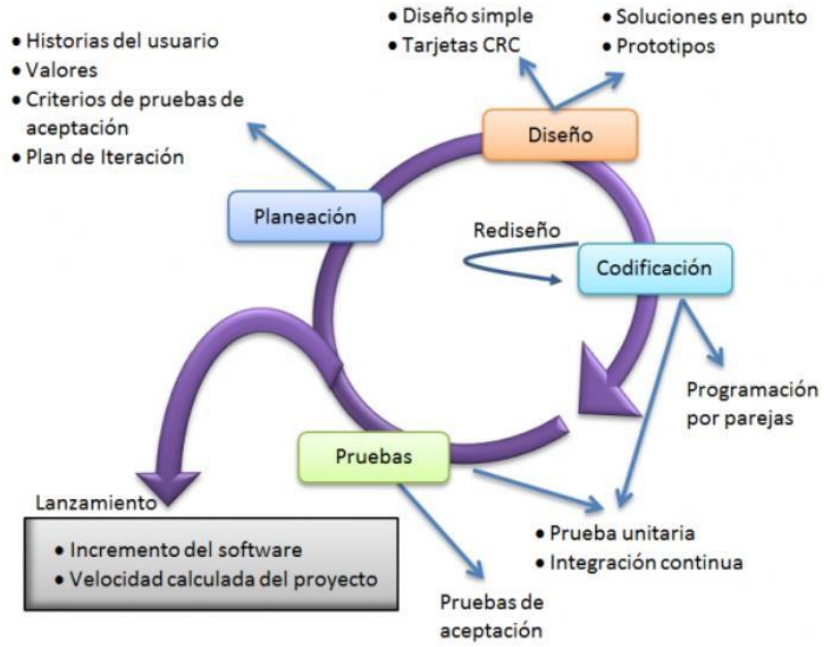

Figura 19. Marco de trabajo de la Metodología XP (Lopez.R.E. 2015)

3.6.3.3 Test Driven Development (TDD)

También conocida como "desarrollo orientado a pruebas", sería como la forma de aplicar el Extrem Programming. Esta técnica, por decirlo de alguna forma, se da en el ciclo de vida de la metodología XP, durante las fases de iteración, producción y mantenimiento. Siendo una técnica orientada al desarrollo de proyectos a través de las pruebas, y debido a su importancia, ya se considera una metodología fuera del Extreme Programming. Como comenta Carvajal (Carvajal Riola 2008), se puede considerar esta forma de desarrollar proyectos una metodología, al presentar normas y formas de desarrollo, que condicionan el desarrollo del proyecto, aumentando su calidad. Es aplicable de forma independiente, o juntamente con otras metodologías como SCRUM, o FDD, o waterfall.

El proceso consiste en iterar 3 procesos que se repiten dentro de esta metodología:

1. Detallar las pruebas para la funcionalidad que se quiere implementar.
2. Codificar dicha funcionalidad hasta que pase la prueba.
3. Recodificar ambos códigos, para estructurarlo.

De esta forma, se itera en bucle, construyendo las funcionalidades del sistema a desarrollar.

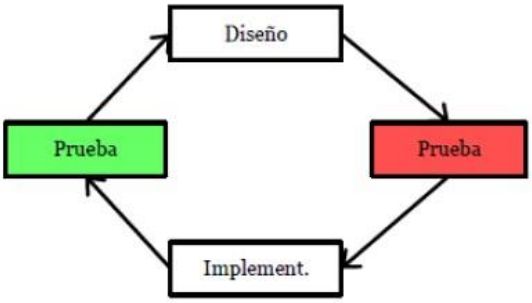

Figura 20. Esquema del proceso TDD. (Gimson 2012)

3.6.3.4 Crystal Methodologies

Metodología basada en la clasificación de la complejidad de un proyecto, en función de una serie de colores, de forma que cuanto más oscuro más complejo será el proyecto. Este método, además propone utilizan un color por proyecto dependiendo del tamaño de este y de su criticidad. Debido a esto, no hay una norma estándar de Crystal, sino que hay una aplicación específica de la metodología para cada proyecto sobre el que se haya aplicado cada tipo de proyecto.

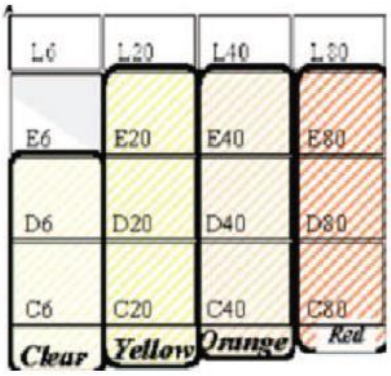

Figura 21. Diagrama de criticidad Crystal (Gimson 2012)

Cada letra de la figura del ejemplo anterior hace referencia a los distintos tipos de riesgos:

- **C**: Pérdida de confort por un fallo en el sistema.
- **D**: Pérdida de dinero (dinero propio).
- **E**: Pérdida de dinero (dinero por préstamos).
- **L**: Pérdida de vidas humanas por un fallo en el sistema.

Los números hacen referencia a las personas que forman el proyecto de la siguiente forma (Gimson 2012):

- *Clear*, para menos de 8 personas.
- *Amarillo*, entre 10 y 20 personas.
- *Naranja* entre 20 y 50 personas.
- *Rojo* entre 50 y 100 personas, siguiendo este patrón con otros colores.

3.6.3.5 Dynamic Systems Development Method (DSDM)

Desarrollo de sistemas centrada exclusivamente en que se satisfagan las necesidades del usuario. Este sistema fue publicado en febrero de 1995, como un método genérico para personas, procesos y herramientas. ***DSDM Consortium*** es la propietaria y administradora del método DSDM y solo sus miembros pueden utilizarlo.

Para el DSDM, el desarrollo se ve como algo iterativo e incremental, en el que el usuario debe estar siempre presente, y en el que hay estar preparado para los cambios, siendo la única metodología que cubre el ciclo de vida del proyecto en su totalidad, al incorporar técnicas de gestión de proyectos, y otras técnicas para garantizar la rentabilidad del proyecto, de acuerdo a la solución prevista.

Las principales características del DSDM son:

1. **Alta velocidad de desarrollo**, adaptabilidad, y entrega de producto sin que merme la tecnología aplicada.
2. Está considerada como una **RAD** (Rapid Aplication Development) siendo idónea para proyectos cuyo plazo de tiempo, márgenes de costes son mínimos.
3. Al tratarse de un **desarrollo por trozos**, y en bucle, tanto usuarios como desarrolladores trabajan codo con codo, dejando de lado la programación de reuniones, pues se encuentran en una "reunión indefinida" al trabajar juntos.

Este método se divide en 5 fases, siendo las 3 últimas iterativas: estudio de viabilidad, estudio del negocio, modelado funcional, diseño y construcción, y finalmente implementación.

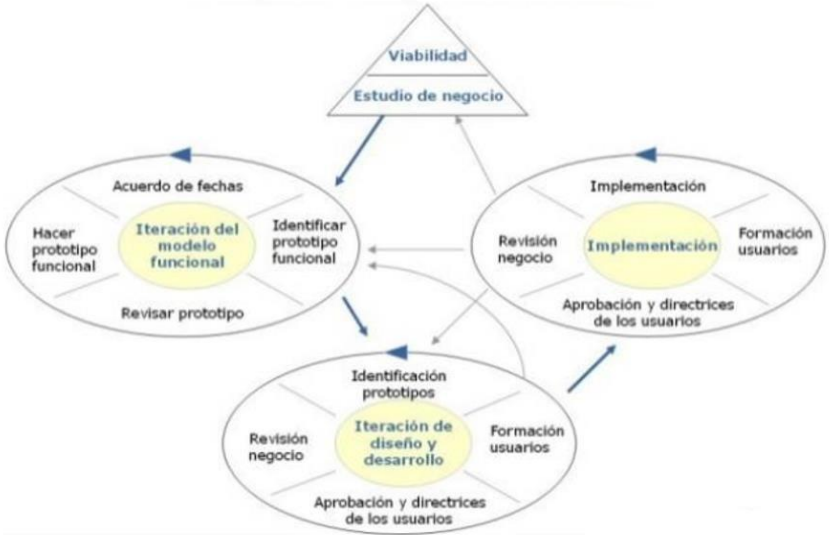

Figura 22. Diagrama del proceso DSDM(Gimson 2012)

3.6.3.6 Adaptive Software Development (ASD)

Fue creada por Jim Highsmith. Sus principales características son:

1. Se trata de un proceso **iterativo**.
2. También está considerada una **RAD** (Rapid Aplication Development) siendo idónea para proyectos cuyo plazo de tiempo, márgenes de costes son mínimos.
3. Se centra en los **componentes software** en lugar de las tareas a desarrollar.
4. Permite **introducir modificaciones**.

Esta metodología propone la ejecución de tres fases: especulación, colaboración y aprendizaje.

Figura 23. Fases Proceso ASD

- **Especulación:** se da comienzo al proyecto planificando las características de lo que se necesita.
- **Colaboración:** se realiza el desarrollo de software del punto anterior.
- **Aprendizaje:** se revisa el software a presentar en términos de calidad, funcionalidad, etc, y se realiza la entrega, o puesta en producción.

Tras estas etapas, la reflexión por parte de todos los miembros permite obtener nuevos aprendizajes, y volver a iniciar el proceso.

3.6.3.7 Feature-Driven Development (FDD)

Se basa en un proceso, creado por Jeff De Luca y Peter Coad en 1997, con **iteraciones relativamente breves, que crean un software funcional que los usuarios pueden ver y probar.**

Cada iteración no debería de extenderse más allá de las 2 semanas, estando centradas las tareas en diseñar y programar el sistema requerido a raíz de un listado de funcionalidades o características que debieran llevar dicho software y que se puedan construir en dicho periodo de tiempo.

Al contrario que el resto de las metodologías, **esta metodología no cubre todo el ciclo de vida**, centrándose en diseñar y construir el software.

La FDD divide el proceso en 5 fases en los que se va diseñando el software que se necesita:

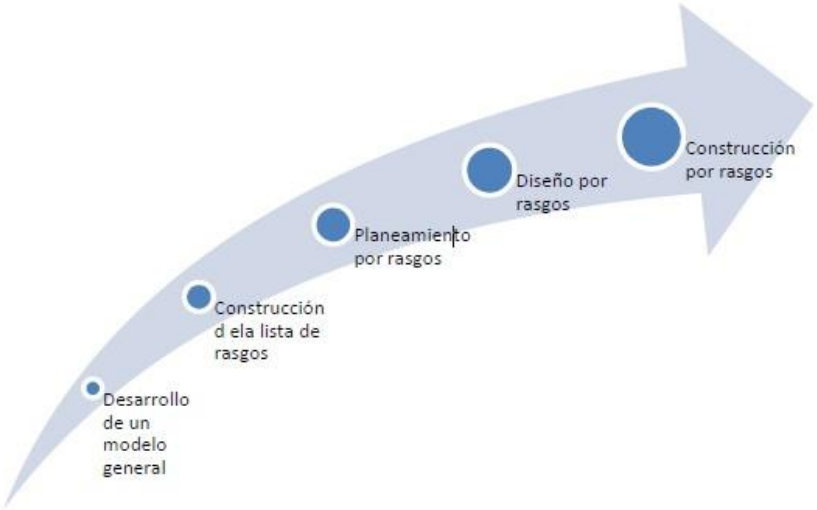

Figura 24. Fases de la Metodología FDD

3.6.3.8 Lean Development (LD)

Fue creada por Bob Charette en 2003, a raíz de los conocimientos adquiridos al trabajar en el sector automovilístico japonés durante la década de los 80, pero se dio a conocer en 2003 mediante el libro "*Lean Software development: An Agile Toolkit*" de los aurores *Mary y Tom Poppendieck*, donde se establecieron las bases de la aplicación de esta metodología.

El LD, se podría definir como el "*Lean Manufacturing*" que se aplicaba durante la época en Toyota, pero aplicado al desarrollo informático. Se basa en los principios Lean considerando los cambios como riesgos, que si se manejan adecuadamente pueden convertirse en oportunidades para mejorar. Su principal característica es el método de implementación de los 7 principios de la filosofía Lean:

1. **Eliminar los residuos.**
2. **Ampliar el aprendizaje.**
3. **Decidir lo más tarde posible.**
4. **Entregar lo antes posible.**
5. **Potenciar el equipo.**
6. **Crear integridad.**
7. **Visión global.**

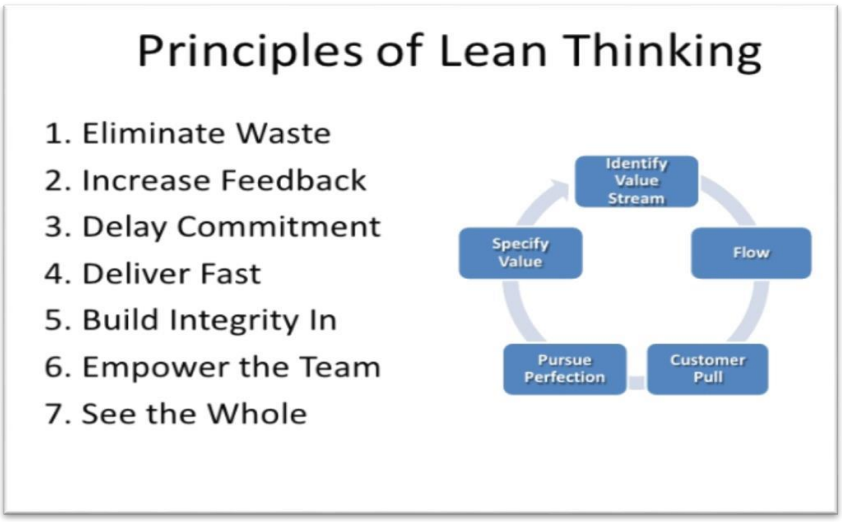

Figura 25. Principios de Lean aplicados al desarrollo

3.7 Metodologías Híbridas

Combinan las mejores prácticas de las metodologías tradicionales y ágiles, incorporando las ventajas de ambas.
El uso de estás metodologías se ha ido incrementando de manera paulatina, ya que se adaptan a la cultura y necesidades de cada organismo, generando un incremento de productividad. Algunos ejemplos de este tipo de metodología son: EssUP (Essencial Unified Process), la combinación de Cascada y Scrum y SXP (Scrum y XP).
Entre los beneficios del uso de la metodología híbrida, podemos encontrar:

- Simple y fácil de entender y usar.

- Mejora continua exponencial.
- Flexibilidad a cambios dentro de un alcance de presupuesto definido.

Figura 26. Metodologías híbridas

Debemos tener en cuenta que en una misma organización pueden convivir más de un tipo de metodología, en base a una serie de criterios, entre los que podemos destacar:

- **Criterios Internos:** el nivel de madurez de la organización, la naturaleza del proyecto, los interesados, los procesos y la tecnología a utilizar.
- **Criterios Externos:** las expectativas y requisitos de los agentes externos que participan en los diferentes procesos de la organización.

3.7.1 Ejemplo Metodología Híbirda: EssUP

3.7.1.1 Introducción

EssUP está basado en casos de uso, CMMI (Capability Maturity Model Integration) y desarrollo ágil. Se considera una mejora sobre RUP (Rational Unified Process) ya que en este último todas las prácticas están relacionadas y no pueden ser usadas de forma aislada. EssUP está soportado por Microsoft Visual Studio Team System, y Eclipse.

El proceso esencial Unificado (EssUP) es un conjunto de prácticas que en conjunto forman el conocimiento esencial de un ciclo de vida de desarrollo de software.

Las prácticas integran principios que se han mostrado efectivos en los campos del proceso unificado, del desarrollo ágil y la madurez de los procesos, haciendo énfasis en las capacidades que ofrecen: estructura, agilidad y mejora de procesos.

Su creador, Ivar Hjalmar Jacobson 2 de septiembre de 1939, es un ingeniero sueco en Ciencias de la computación. Inventó el diagrama de secuencia y desarrolló los diagramas de colaboración. También impuso el uso de diagramas de estado de transición para describir los flujos de mensajes entre los componentes. Fue uno de los desarrolladores originales del SDL (lenguaje de especificación), que se convirtió en estándar en 1967.

En noviembre de 2005, Jacobson anunció la Essential Unified Process (EssUP), una nueva "práctica" centrada en el proceso de desarrollo de software. Se trata de un nuevo comienzo a la integración de prácticas eficaces de entre los tres principales campos de proceso: el proceso unificado, los métodos ágiles y el proceso de madurez. Cada uno de ellos contribuye diferentes capacidades: estructura, la agilidad y la mejora de procesos.

3.7.1.2 Características

EssUP identifica prácticas como desarrollo en base a casos de uso, desarrollo iterativo, architecture driven development, prácticas para el equipo y prácticas para los procesos, que son tomados de RUP, CMMI, y desarrollo ágil.

La metodología denomina Essential Practicess al conjunto de prácticas, las cuales esta divididas en 6 prácticas técnicas y 6 prácticas transversales.

Prácticas Técnicas

1. **Architecture Essentials**

Esta práctica está destinada a obtener una base firme para el desarrollo de sistemas de alta calidad y robustos.

A partir de aquí la arquitectura evoluciona en función de nuevos requisitos y resultados de las pruebas. Todos los sistemas construidos están sujetos al cumplimiento de los test para asegurar la validez de la implementación de la arquitectura.

2. **Iterative Essentials**

Esta práctica reduce el riesgo del proyecto mediante el desarrollo incremental del sistema sobre un número de iteraciones. El proyecto se descompone en trozos más pequeños o mini proyectos, independientes y limitados en el tiempo. Esta práctica permite al equipo:

3. **Use-Case Essentials**

Una forma ágil, sencilla de controlar y de seguir el desarrollo del proyecto software.

4. **Component Essentials**

Esta práctica se utiliza para desarrollar complejos sistemas formados de componentes más pequeños y más simples.

5. **Product Essentials**

Administrando versiones del producto Esta práctica se utiliza para administrar el desarrollo de sucesivas evoluciones de un sistema software como una serie de versiones del producto.

6. **Bussiness Use Case Essentials**

Caso de uso de negocio: gestionando tus requisitos de negocio. Podremos captar y redefinir tal y como requieren tus procesos de negocio para que se llegue a una solución software más eficaz y que mejore las prestaciones y el valor que te aportará. Con la práctica de caso de uso de negocio esencial, tu equipo se enriquecerá al comprender las necesidades del negocio, y cómo involucrar al negocio para lograr cubrir las necesidades de la organización.

Prácticas Transversales

1. **Unified Process Lifecycle Essentials**

Esta práctica se utiliza para establecer un control sobre el ciclo de vida de un proyecto de desarrollo iterativo

2. **The Common Milestones**

Este patrón define un conjunto de hitos o puntos de paso, apropiado para la planificación y el seguimiento de todas las variantes de desarrollo iterativo e incremental de los proyectos software.

3. **The Lifecycle Phases**

Esta práctica perfecciona el patrón Common Milestones mediante la definición de cuatro fases para el adecuado progreso del proyecto hacia el éxito a través de los tres milestones anteriores. El proyecto o ciclo de lanzamiento del producto/versión se divide en cuatro fases secuenciales, cada una de las cuales tiene objetivos bien definidos:

- Inicio (Inception) - Confirmación del alcance y objetivos y control de los riesgos asociados al negocio.
- Elaboración (Elaboration) - Concreción de los planes y control de los riesgos arquitectónicos y técnicos.
- Construcción (Construction) - Construcción del producto y control de los riesgos asociados a la ejecución del proyecto.
- Transición (Transition) - Entrega del producto y control de los riesgos de despliegue.

Estas cuatro fases son tomadas de RUP por lo que en dicha metodología se puede encontrar información ampliada y actividades concretas a realizar en cada fase.

4. **Team Essentials**

Esta práctica es utilizada para reunir un equipo de proyecto y establecer un ambiente de trabajo efectivo. Para ello el equipo debe:

- Adoptar las medidas de liderazgo y de organización.
- Establecer y adquirir las competencias necesarias para tener éxito.
- Desarrollar formas efectivas de colaborar y organizar su trabajo.
- Crear una ambiente en el que todos los miembros del equipo sean capaces de contribuir al máximo de su capacidad durante todo el proyecto.

5. **Process Essentials**

Está práctica permite mejorar y adaptar la forma de trabajo del equipo, en concreto, permite al equipo:

- Identificar, preparar y reunir un conjunto de prácticas y herramientas adecuadas para conseguir los objetivos del proyecto.
- Introducir nuevas prácticas individual y gradualmente según sea necesario.
- Comparar e integrar prácticas estándar y particulares preservando los aspectos que el equipo ejecuta correctamente y señalando aquellos que no.
- Evolucionar las prácticas en base a la experiencia y las lecciones aprendidas.

6. **Modeling Essentials**

Esta práctica se utiliza para establecer un estilo y tipo apropiados para guiar las actividades de desarrollo. Con los modelos podremos:

- Comunicar los requisitos, estructura y comportamiento del sistema.
- Ver diferentes perspectivas del sistema y entender cómo se relacionan entre sí.
- Emplear los modelos adecuados que mejor se adaptan a cada necesidad.
- Ser ágil en la forma de abordar el modelado y documentación.
- Concentrarse en lo esencial para evitar la producción de documentación innecesaria.

4 Ejemplo de metodologías ágiles

En esta sección se hace una introducción sobre el sistema **Kanban**, y su uso en el desarrollo de proyectos IT, para continuar abordando la metodología **Scrum**: su origen, elementos principales de metodología y herramientas relacionadas con el uso de dicha metodología.

4.1 Kanban

El Sistema Kanban es considerado como el uso de *Lean* al desarrollo de software, siendo este último, quien sirvió de inspiración al *Kanban aplicado a la ingeniería de software*, gracias entre otras personas a David Anderson, Arlo Belshee y Kenji Hiranabe, y a los ya citados autores *Mary y Tom Poppendieck* que con su libro "*Lean Software development: An Agile Toolkit*" sentaron las bases de esta herramienta.

Kanban postula que el trabajo en curso (WIP) debería restringirse, de forma que única y exclusivamente comience una tarea, o trabajo cuando la tarea anterior haya sido realizada, finalizada o haya avanzado a un estado posterior.

El Kanban (también conocido por utilizarse "tarjetas señalizadoras") indica mediante, en el caso del software mediante una tarjeta, que existen nuevas tareas que pueden ser llevadas a cabo porque las actuales ya han sido acabadas o no pueden seguir avanzando. Según David J. Anderson (Anderson 2010), aunque el uso de Kanban pueda parecer sutil, es capaz de modificar todos los aspectos que rodean a una empresa.

4.1.1 Objetivos

Los objetivos de Kanban son:

- Equilibrar demanda de trabajo con la capacidad de trabajo.
- Limitar el "work in progress", mejorando el "flujo" de trabajo, descubriendo los problemas de forma anticipada, logrando un ritmo constante.
- Controlar el trabajo (en lugar de las personas) coordinándolo, de forma que se puedan encontrar los encolamientos se puedan tomar decisiones que aporten valor.
- Equipos autogestionados.
- Implementar una cultura de mejora continua.

4.1.2 Ventajas

Los beneficios que aporta Kanban al desarrollo de proyectos son:

- Ajuste de cada proceso y flujo de valor, a medida.
- Simplificación de reglas para optimizar el trabajo en función del valor que genera.
- Mejor gestión de riesgos.
- Tolerancia a la experimentación,
- Mayor colaboración dentro y fuera del equipo.
- Mejor calidad de trabajo.
- Ritmo de trabajo constante.

4.1.3 Sistema

Los sistemas Kanban constituyen una herramienta para la planificación de los trabajos. En contra de las metodologías de proyectos ágiles estándar, que utilizan periodos de tiempo prefijados (conocidas como *TimeBox*) acordadas al inicio de los mismos, en los que al final de cada iteración se entrega una o varias historias de usuarios[ii], los sistemas Kanban se centran en un flujo continuo de trabajo, de forma que los equipos de desarrollo van trabajando de forma continua, afrontando cada historia de usuario, desarrollándola y entregándola en cuanto esté finalizada, siempre de forma individual.

El proceso del estado del trabajo se entiende a través de un mapa de flujo de valor donde se acuerdan los límites del *"work in progress"* y a través del cual, se comienza a trabajar empujando las distintas tareas a través de las distintas señales, o tarjetas Kanban.

4.1.3.1 Flujo de Valor

El flujo de valor de un proyecto se compone de todos aquellos procesos y subprocesos que, realizados, o en fase de realización, van desde la planificación hasta la entrega y agregan valor al producto.

Figura 27. Evolución del flujo de valor en el desarrollo de un proyecto.

Cada proceso (o columna), está compuesta de dos columnas más: una para las tareas en curso (WIP), y otra para las tareas terminadas. Los números encima de cada columna representan la máxima cantidad de tareas que puede estar realizándose de manera simultánea en cada proceso.

4.1.3.2 Límites en Kanban

En el sistema Kanban, es importante poner límites en cada proceso y subproceso. Esto es debido a que si se limita una carga de trabajo a una capacidad asignada, dicha calidad aumenta reduciendo los costes, aumentando también la velocidad y disminuyendo los retrabajos. Esto se debe a 3 motivos (Gimson 2012):

- La velocidad de las tareas aumenta cuanto menor es la cantidad de tareas por hacer (para una misma etapa.
- El hecho de trabajar en varias tareas a la vez, provoca que una pérdida de tiempo y de concentración entre las tareas en las que se está trabajando.

- Cuanto antes se finalice una tarea, antes se aprenderá de los errores de la misma, y se descubrirán los problemas con antelación.

Otro detalle importante, es el hecho de que la cantidad de trabajo es limitada. Esto se debe a que, debido a la variabilidad de las necesidades del cliente, cuanta mayor cantidad de trabajo se asocie, mayor será la dificultad de cambiar las necesidades, si el cliente cambia también de prioridad u objetivo. En este sentido, en el ámbito de desarrollo de proyectos de software, es más cómodo trabajar no con la cantidad de tareas, si no como la suma de "story points" o equivalente (talla de camisetas etc.)

4.1.3.3 Tablero Kanban

La aplicación de Kanban al desarrollo de proyectos, puede verse de una forma rápida gracias a la agrupación de las tareas en un tablero, en el que los "post-its" que representan dichas tareas, de forma que sea visible y de un vistazo sea posible saber en qué está trabajando cada miembro del grupo, y la carga de trabajo a la que se está viendo sometido cada uno, de forma que el ritmo de trabajo sea unísono.

Este tablero, se suele colocar en una zona visible dentro del espacio de trabajo, de forma que todo el mundo pueda verlo sin problemas, y debe ser continuo, de forma que las tareas no vayan acumulando "post-its" hasta que la tarea haya sido terminada, sino que conforme se vaya ejecutando una tarea, las nuevas tareas relacionadas con esta última (fallos, mejoras, o nuevas tareas), se dispongan al comienzo del tablero, de forma que en la revisiones con los usuarios, se prioricen las tareas que le sean prioritarias.

Un tablero Kanban, ha de tener toda la información necesaria, para que el grupo de trabajo, sea capaz de decidir sin tener que consultar o sin ser supervisados.

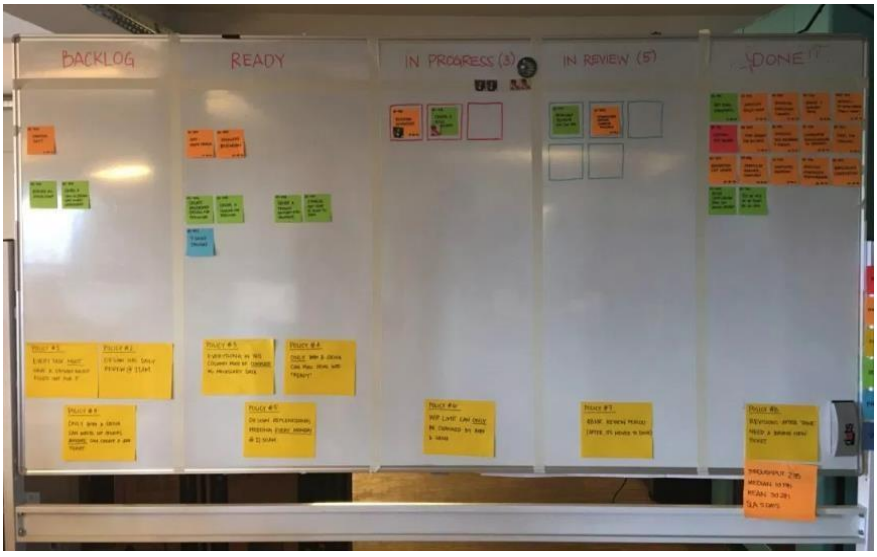

Figura 16. Tablero Kanban

En cada equipo o grupo de trabajo, las **normas** se definen dentro del mismo, pudiendo establecerse cuantos criterios necesite el equipo para poder trabajar, como por ejemplo:

- Las tarjetas verdes representan tareas completadas.
- Las tarjetas rojas representan tareas bloqueadas por otras, o limitaciones encontradas durante el proceso.
- Las tarjetas blancas representan productos en proceso para realizarse.

- Las tarjetas amarillas representan tareas a realizar para terminar un producto o subproducto.
- Existen también reglas de decisión que indican el procedimiento a la hora de modificar las tareas en el tablero, pero también pueden contener información relevante para el proceso, notas, “feedback”, que se suelen colocar en la parte inferior de cada proceso del tablero.
- Cada columna representa un proceso, con la marca del límite de tareas máximas simultáneas.
- Los avatares representan a los responsables de cada tarea. No se suele nombrar al responsable de cada tarjeta porque puede ser tratada por diferentes personas en diferentes momentos.

Cada **tarjeta** puede tener los siguientes datos:

1. **Fecha** de la tarea y de entrada y salida de cada proceso.
2. **Fecha comprometida** (si aplica).
3. **Descripción**.
4. **Prioridad**.
5. **Estimación** de la duración.

4.1.3.4 Indicadores

Kanban dispone de varios indicadores como pudiera ser el límite de trabajo, que permite decidir si se debe comenzar un nuevo trabajo. De esta forma, y teniendo presente cuándo se plantearon estos límites se pueden regular los problemas que surgen en los procesos, así como liberar parte de recursos asignados a otras tareas, si fuera necesario.

Además, las colas de tareas por hacer, y en los propios procesos, no dan pie a malas interpretaciones, porque todo el mundo puede ver en qué tarea está trabajando cada miembro, y con qué prioridad.

4.1.3.5 Funcionamiento y problemas en Kanban

Una vez que comienza el equipo a trabajar, se suele lograr un ritmo constante, equilibrado y sostenido en el tiempo conforme las tareas se van realizando, y cada miembro se habitúa a esta nueva forma de trabajar.

Una de las situaciones que se deben evitar es la generación de cuellos de botella, ya que además de parar el ritmo de trabajo, son origen de errores, fallos, y olvidos que se pueden, y deben evitar. Esto puede ser debido a una mala definición de límites, problema de limitación de recursos, o una oportunidad de mejora en otros procesos. En caso de producirse, lo mejor es usar recursos libres para desatascar las colas, y examinar el proceso a posteriori para buscar mejoras.

Otra situación que puede darse es la falta de asignación de tareas, porque una persona haya completado todas sus tareas, pero no pueda proseguir porque la tarea con la que comenzar, todavía no la ha terminado la persona de la que depende en ese momento dicha tarea. De nuevo, puede deberse a un problema de definición de límites, recursos, y una oportunidad de mejora.

Una situación adicional que se puede presentar se da cuando un recurso asignado a una tarea se libera y otro miembro aún no ha finalizado su tarea. En estos casos, la persona que no tengan

tareas asignadas en ese momento podría asignarse otra tarea de la cola de tareas o ayudar al compañero a terminar la suya. La regla no escrita, dice que un recurso libre debería ayudar al compañero a terminar el trabajo siempre que sea posible. De esta forma cuando los dos finalizan dicha tarea, pueden asignarse dos tareas de la cola de tareas.

4.2 Scrum

4.2.1 Historia de Scrum

Scrum forma parte de una evolución del modelo *Shashimi*, que fue concebido en Japón como una variante a la metodología waterfall, que buscaba una mayor velocidad, flexibilidad, y un mayor feedback al final de cada etapa del modelo en cascada.

La primera vez que apareció el término *Scrum* fue en el año 1986, de la mano de Hirotaka Takeuchi e Ikujiro Nonaka en *The New New Product Development Game* (Takeuchi y Nonaka 1986) donde se hizo una recopilación de nuevos métodos de desarrollo exitosos, utilizados en empresas de Japón y en Estados Unidos (como por ej. las cámaras fotográficas Canon, fotocopiadoras de Xerox, vehículos Honda, los ordenadores de Hp, etc…). Los equipos detrás de estos productos fueron capaces de lanzar al mercado, en mucho menos tiempo que la competencia, partiendo de requisitos muy generalistas, de funcionalidades, y, por tanto, productos muy novedosos, compartiendo ciertos patrones en el desarrollo de proyectos. En dicho estudio presentado por los autores anteriormente citados, se realizó una analogía entre la manera de trabajar entre los equipos de desarrolladores, y la colaboración entre los jugadores de un equipo de rugby. La palabra Scrum tiene el mismo origen que en el deporte del rugby, donde se utiliza en la jugada en la que se debe devolver el balón que ha salido del campo de forma colectiva (Scrum significa melé en castellano).

Scrum no fue puesto en práctica hasta 1993, cuando Jeff Sutherland aplicó el modelo al desarrollo informático para la compañia *Easel Corporation*. En 1996, Jeff Sutherland, y Ken Schwaber presentaron las normas que usaron para los desarrollos informáticos en OOPSLA 95 en Austin, Texas (Conferencia Internacional sobre programación, lenguajes y aplicaciones que es realizada cada año por la A.C.M. (Association for Computing Machinery))

Posteriormente, Schwaber y Mike Beedle presentaron SCRUM en 2001 mediante el libro *Agile Software Development with Scrum*.

Actualmente se encuentra publicada la Guía de Scrum (*The Scrum Guide*), como la referencia oficial de normas de Scrum, escrita por Ken Schwaber y Jeff Sutherland.

La última versión de esta guía fue publicada en noviembre de 2017.

4.2.2 Descripción general de Scrum.

Según la Guía del Conocimiento de Scrum (Satpathy 2017) un proyecto Scrum consiste en "*un esfuerzo de colaboración para crear un nuevo producto, servicio u otro resultado tal como se define en la declaración de la visión del proyecto" (Project Vision Statement)*.

Los proyectos suelen estar perjudicados por motivos de tiempo, coste, alcance, calidad, recursos, y demás restricciones que dificultan la planificación, y ejecución de estos. No obstante, la finalización de un proyecto en tiempo y forma con éxito repercute de forma económica a la

empresa. Por ello, es necesario que las empresas sean capaces de implantar una metodología capaz de gestionar todos estos aspectos de forma adecuada.

Scrum es un "*framework*" adaptable, rápido, iterativo, eficaz, y flexible concebido para dar valor a un proyecto de una manera ágil y rápida. Scrum permite mejorar la comunicación y crear un entorno de responsabilidad común entre los participantes del proyecto, y una tasa de avance de forma constante.

Scrum, por cómo está definido, es compatible con todo tipo de productos y todo tipo de proyectos, independientemente de la complejidad.

El desarrollo de Scrum, se fragmenta en pequeños periodos de tiempo de trabajo, en los que se trabaja de una forma concentrada, llamados Sprints. Estos sprints, permiten un alto ritmo de trabajo, en parte debido a que los equipo son altamente cualificados, y con las autorizaciones suficientes como para tomar decisiones relativamente importantes sobre la marcha, eliminando tiempos de espera que no aportan nada.

Scrum se basa en tres pilares fundamentales:

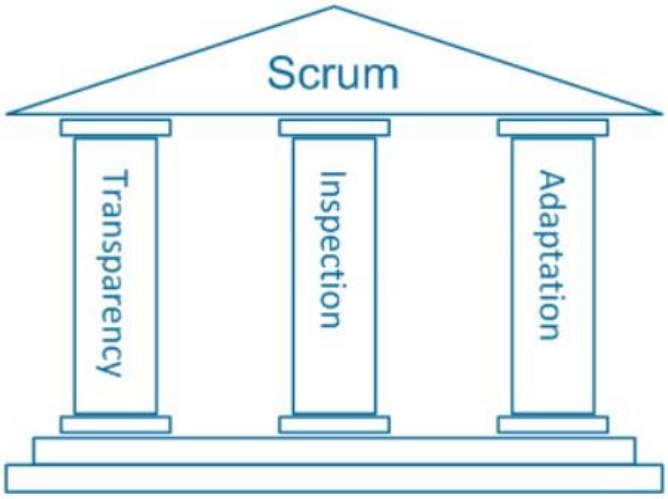

Figura 17. Pilares de la Metodología Scrum

4.2.3 Roles de Scrum.

En Scrum, existen una serie de roles que se aglutinan en 2 conjuntos de personas: de una parte, estarían las personas que deben estar involucradas tanto en el proyecto como en Scrum, y de otra parte, están las personas que no siendo imprescindibles para el desarrollo del proyecto, o la aplicación de Scrum, su presencia en determinadas situaciones es necesaria.

Estos 2 grupos, también son conocidos como el grupo de los *cerdos* y el de las *gallinas*, por una viñeta muy conocida en relación con un cerdo y una gallina, y la intención de montar un restaurante entre ambos:

Roles Comprometidos

- **Product Owner**: Representa la voz del cliente, toma las decisiones, y es la que conoce realmente las necesidades del producto como usuario. Escribe las ***historias de usuario***, y las prioriza en el ***Product Backlog***.

- **Scrum Master**: Es la persona encargada de garantizar que se aplica la metodología. Se encarga también de que el equipo no se desvíe de sus tareas, y trata de resolver aquellos problemas que impidan la consecución del ***sprint***. Realizando tareas de coordinación, dirige los ***scrums diarios***, y sigue el avance del proyecto.
- **Equipo**: Está compuesto por menos de 8 personas (en caso de haber más, se dividiría en más equipos que trabajarían sobre el producto backlog del mismo). Son responsables de desarrollar y presentar el producto, tomando decisiones, pues tienen asumidas responsabilidades de decisión y de gestión. Son capaces de estimar el esfuerzo de las tareas del *product backlog*, y entregar un grupo de tareas del producto Backlog al final del Sprint.

Roles Involucrados

- **Usuarios**: personas para las que se diseña el producto.
- **Stakeholders o Interesados (Clientes, Proveedores)**: personas que, de forma indirecta, están relacionados con el proyecto porque de alguna forma les afecta (ya sea en positivo o en negativo). Participan durante la revisión de cada **Sprint.**
- **Managers**: Son aquellas personas encargadas de preparar el marco de trabajo en el que se desarollará el producto. Además, participan en la toma de decisiones en situaciones bloqueantes o de atasco.

4.2.4 Elementos de Scrum

Los principales elementos de Scrum que son necesario conocer para entender el proceso, si bien se utilizarán otros que se irán describiendo más adelante, son (José, H., Canós, P. L., & Carmen 2003):

4.2.4.1 Product Backlog

El Product Backlog consiste en una lista de funcionalidades necesarias para negocio priorizada en función de las necesidades. Estas funcionalidades están basadas en descripciones a muy alto nivel de lo que se espera que realice el producto, en el que se detallan de una forma resumida, las características, funciones, mejoras que deberían tener el producto. Esta lista, supone el comienzo de la recopilación de los requisitos del producto, y de los nuevos que irá adquiriendo el producto en su evolución.

El Product Backlog debe ser gestionado por el negocio, que, con la ayuda del Scrum Master, le dará una idea del coste estimado, y contendrá todo lo que aporte un valor final al producto.

Las principales características son:

- Debe incluir todas las **necesidades** del producto, expresados mediante historias de usuario.
- Para cada necesidad, se incluirá el **valor** que aporta al negocio y un **coste** estimado, de forma que puedan ser priorizadas basándose en el ROI[iii].
- Se tendrán que indicar el número de **sprints** o iteraciones, y los despliegues acordados.
- Se han de incluir los **riesgos** que puedan contemplarse, y cómo se van a mitigar.

Antes de comenzar a desarrollar deben definirse los objetivos, y las necesidades del producto. Sin tener que entrar en mucho detalle, basta con que tenga el contenido suficiente para poder empezar a trabajar.

La aplicación de este conjunto de características hará que:

- El proyecto no se detendrá ante necesidades poco claras, permitiendo obtener los resultados sobre los que se está trabajando desde el principio, antes.
- El resto de las necesidades irán surgiendo a lo largo del avance del desarrollo, no perdiendo tiempo en analizar dichas necesidades al comienzo.
- Estas necesidades secundarias, es posible que no sean necesarias al estar ya cubiertas, porque hayan surgido nuevas más interesantes, o que se descarten por su coste. Una vez definidos los requisitos se tendrá que acordar cuándo se tiene que entender un objetivo como terminado o completado.

Un producto o entrega, se dará por finalizado si:

- Se puede generar un documento que contenga una demostración con todas las necesidades que se pedían al comienzo, cubiertas y operativas.
- Contiene todo aquello que se está realizando sobre el producto y que así ha sido solicitado por el cliente.

Adicionalmente a la determinación de una funcionalidad completada, se deberían establecer una serie de criterios o condiciones que definan cuándo, o en qué momento se dará por buena la prueba, o la aceptación de las funcionalidades que se solicitaban al comienzo de la iteración.

Todos estos aspectos, harán que el Product Backlog vaya evolucionando con el proyecto, dando por completadas funcionalidades, surgiendo nuevas, descartando otras existentes, o re-priorizando las necesidades que queden por desarrollar.

4.2.4.2 Historias de Usuario.

Las historias de usuario son el conjunto de funcionalidades necesarias para el negocio, que necesita que sean implementadas en el desarrollo software. Lo lógico es que sean diseñadas de forman común entre usuarios y resto del equipo, de forma que irán variando a lo largo del desarrollo.

Cada historia de usuario cumple una serie de características:

- **Card:** breve descripción a modo de recordatorio de la historia.
- **Conversation:** Pequeño resumen que garantiza que todos han entendido lo que se necesita y lo que quiere llevar a cabo.
- **Confirmation:** casos de prueba que describirán si la funcionalidad cubre lo que se necesita

En cuanto al modelo, existen varios tipos, pero debería contener:

- **ID**: Identificador de la historia de usuario.
- **Título**: breve descripción de la historia.
- **Estimación**: Medida del esfuerzo (criterio establecido al comenzar el proyecto) para implantar la necesidad.
- **Prioridad**: Rango de prioridad respecto al resto de historias, siendo más prioritario cuanto mayor sea dicho valor. Otra aproximación a la priorización se realiza mediante algún método de priorización, como el método **Moscow:**
    - **Must**: Es necesario terminar el requisito para finalizar el proyecto.
    - **Should**: Es necesario completar la funcionalidad como sea, pero el éxito del proyecto no depende de ella.

    - **Could**: Se podría completar esta funcionalidad, si la implementación de esta no afecta a la consecución de los objetivos principales.
    - **Would**: Se podría llevar a cabo la funcionalidad si sobrase tiempo.
- **Dependencias:** Una historia de usuario debería ser independiente, pero no siempre es así.

4.2.4.3 Sprint Backlog

Está compuesto por el conjunto de tareas que componen un sprint. Mediante este, se identifican y definen todas las tareas necesarias para completar el sprint.

Mientras que en el Sprint Backlog aparecen las distintas funcionalidades que se necesitan, por otro lado, también aparecen todas las tareas necesarias para completar dichas funcionalidades, de forma que el Sprint Backlog es una parte del Product Backlog (como un subconjunto).

También al estar las historias de usuario ya priorizadas, las tareas asociadas a cada historia, ya vienen priorizadas por la propia historia de usuario que fue ordenada durante el product backlog.

Sin embargo, el hecho de que puedan existir tareas que no se hayan terminado en un anterior sprint, harán que, en el comienzo del siguiente, se acumulen dichas tareas.

Todo sprint backlog debe ser aceptado por el equipo del proyecto (sobretodo por los programadores), quedando a criterio del equipo la posibilidad de modificarlo.

En caso de necesitar añadir nuevas funcionalidades de forma urgente, deberán ser incluidas en el producto backlog, y tenidas en cuenta para el siguiente sprint.

Las características son:

- Es un listado de necesidades para el usuario ordenado por prioridad. o Podrían existir dependencias entre ellas, y en ese caso habría que reflejarlo.
- Las tareas deben tener un esfuerzo estimado entre 4 a 16 horas.

El formato posible para hacer estas listas es variado:

- Hojas de Cálculo. o Pizarras.
- Herramientas Colaborativas (Trello, Microsoft Teams, …)

Estas tareas se gestionan mediante un *Scrum Taskboard*, donde cada objetivo tiene asociadas las tareas necesarias para completarlo mediante post-its que van cambiando de columna en función del su estado de avance.

Dicho tablero cumple las siguientes características:

- Contiene las tareas a llevar cabo.
- Incluye a la persona que lleva la tarea, estado de esta, y plazo para finalizarla.
- Permite la consulta diaria por sus miembros.
- Permite ver cada día lo que le resta a cada tarea.

4.2.4.4 Incremento

Un incremento se define como la suma de todos los ítems completados en el Sprint backlog. Si hubiera ítems incompletos, deben ser incluidos en el Product backlog para que sean realizados en el próximo sprint. Un ítem se dará por finalizado si este es funcional. El sumatorio de todos ellos debería de dar como resultado el producto final, preparado para ser puesto en producción o desplegado, tras la *Revisión del Sprint* con algunas tareas, aunque a veces se necesiten acciones adicionales (pruebas o documentos)

4.2.5 El proceso de Scrum

Scrum establece un marco para la gestión y desarrollo (entendiendo desarrollo como toma de requisitos, la fase de diseño funcional, y codificación posterior) de proyectos, en el que se definen una serie de iteraciones de una duración máxima de 4 semanas, conocidas como ***sprint***.

Cada ***sprint***, estaría compuesto por 3 fases:

1. **Pre-sprint**: planificación de lo que se va a llevar a cabo en el sprint.
2. **Sprint**: periodo de trabajo.
3. **Post-sprint**: revisión y retrospectiva del sprint.

El ciclo de Scrum se podría resumir en:

Figura 28. Ciclo de vida de un proyecto Scrum. Fuente: northware.mx

4.2.5.1 Planificación Pre-Sprint.

También llamado **sprint 0**, es la primera fase en la que se escuchan las necesidades del usuario, con el objetivo de que las posteriores decisiones que se realicen en las siguientes etapas, agreguen siempre valor.

Para este sprint, el *product owner* debería tener ya elaborado y priorizado un Product Backlog. En dicha reunión, en la que se encuentran el product owner, el equipo de programadores, y el scrum master, el dueño del producto define las funcionalidades que se deberían incluir en el sprint y se definen las tareas necesarias para llevar a cabo dichas funcionalidades. Generalmente, en esta etapa se producen errores en la estimación, ya que se hacen por encima, por lo que es aconsejable no parar mucho en esto, e invertir dicho tiempo en otros aspectos. Además, el equipo determina la lista de tareas y estima cuánto puede cumplirse para ese sprint. Las tareas que se realizan en este sprint 0 son:

- **Definir el objetivo**: se indican los objetivos del proyecto a alto nivel de forma que los componentes del squad, puedan comprender las necesidades. El objetivo del sprint debe ser posible aunque haya tareas incompletas o abandonadas: "*La razón del objetivo de un sprint es dar al equipo cierta flexibilidad respecto a la funcionalidad*" tal como Ken Schwaber y Mike Beedle lo enuncian (Schwaber y Beedle 2001). El objetivo sirve para no perder el foco en las tareas que deben ser llevadas a cabo como principales. Además, el objetivo sirve para que el equipo se se encuentre animado y sean capaces de integrar cualquier circunstancia que pueda darse mientras dura el sprint.
- **Definir "terminado":** esto permitirá saber cuando se ha finalizado una tarea.
- **Definición del Backlog inicial**: Esto permitirá que el sprint que se comience contenga una cantidad suficiente de tareas para llevar a cabo. Dicha lista, como ya comentamos anteriormente, debe ser priorizada por el product owner, de forma que conforme se vayan realizando sprints, se vayan añadiendo nuevas funcionalidades al producto de forma que se vayan cumpliendo las expectativas del usuario.
- **Definición de los entregables**: Aunque anteriormente se haya comentado que la entrega de documentación pasa a un segundo plano con Scrum, sí que es cierto, que sería bueno para el proyecto, establecer cuándo se van a realizar estos documentos para añadir valor al producto, y poder tener un mayor feedback.

Este plan de entregas puede variar por diversos motivos como:

- Cambios en las necesidades, o nuevas necesidades para los usuarios.
- Aparición de nuevas funcionalidades que añaden valor.
- Cambios en el entregable.

Este plan lo debe definir el usuario, que debe determinar las funcionalidades y el coste asociado que asumirá. También, dicho entregable estará condicionado por el equipo de trabajo debido a:

- El tiempo necesario para desarrollarlo.
- La estimación de tiempo y coste.
- Selección de tareas del Backlog del producto a asumir.

Una vez planificado el sprint inicial, se realizará una reunión con todas las personas del equipo para:

- Dimensionar el proyecto.
- Revisar el product backlog.
- Establecer el horario de trabajo, y las próximas reuniones.

Tras esto, el equipo en su conjunto comenzará a trabajar hasta ajustar los recursos con la cantidad de trabajo planteada para el primer sprint, teniendo definido el **Sprint Backlog**.

4.2.5.2 Estimaciones del Sprint Backlog

Para poder comenzar a planificar en la primera reunión de planificación, el equipo de programadores debe saber cuáles son sus capacidades de trabajo, y a qué ritmo puede ir realizando las distintas tareas. Para ello, primeramente, se han de decidir qué historias se incluirán en la pila de historias de usuario del primer sprint.

La forma para decidir qué historias incluir, puede realizarse de dos formas:

1. **Mediante tanteo:**

   En esta situación, los miembros analizan la cantidad historias de usuarios hasta alcanzar un consenso. Este método es útil en sprints cortos.

2. **Mediante estimaciones de velocidad de trabajo:**

   Este método se realiza en 2 fases:

   - Elegir una velocidad estimada.
   - Determinar la cantidad de historias realizables sin alcanzar la velocidad estimada.

La forma óptima de estimar sería revisando el historial del equipo basándonos en sprints anteriores que pudieran dar una idea de la velocidad del equipo.

Una de las formas para realizar esta técnica, sería utilizando un equipo que tenga un mismo historial, de forma que los sprints que se ejecuten sean similares a los ya realizados anteriormente por los miembros en equipos anteriores, y que en este compartan dimensionamiento y condiciones.

Otra forma de realizar la técnica sería mediante el cálculo de recursos:

3. Cálculo del número de días - hombre disponibles (aunque el dato sea poco significativo, porque influyen factores externos que pudieran alterar la dedicación).

4. La velocidad estimada se calcula como:

$$\boldsymbol{Velocidad\ Estimada} = (Dias - hombre\ disponibles) * (factor\ dedicación)$$

5. El factor de dedicación se estima en función del estado del equipo, siendo razonable, estimarlo en función del estado de los últimos sprints.

$$\boldsymbol{Factor\ dedicación} = \frac{(Velocidad\ Real)}{(Dias - hombre\ disponibles)}$$

La velocidad real sería el sumatorio de las estimaciones que se llevaron a cabo para el último sprint.

Para equipos nuevos, puede tratar de compararse con equipos similares, o usar un valor inicial para el factor de dedicación. Se suele recomendar el 70%.

4.2.5.3 Planificación del Sprint.

Conocido como **"Sprint Planning Meeting"**, se trata de una reunión entre product owner, scrum master, y equipo, para elegir que funcionalidades del product backlog se realizarán en el sprint.

Para esta reunión, como comentamos anteriormente, el product owner debe de tener ya realizada su lista de funcionalidades del producto priorizada, de forma que en la reunión se trate de ver, cuáles se pueden aglutinar en un sprint.

El *time-box* de esta reunión es de 8 horas, divididas en 2 partes de 4 horas:

Primera parte de la reunión:

- Se eligen las historias que pasarán a ser entregables.
- El equipo puede opinar sobre las historias, y de su capacidad para llevarlas a cabo en el sprint, pero será el product owner el que tendrá la última palabra.
- El equipo decidirá qué podrá implantar, de todo lo seleccionado por el Product Owner para el sprint.

Segunda parte de la reunión:

- El equipo de desarrollo consultará todas las dudas que pueda tener sobre el Product Backlog al Product Owner.
- El equipo transformará las necesidades o historias en un entregable.

Como fruto de la reunión, se obtendrá, el "**Sprint Backlog**" con todo lo necesario (tareas, especificaciones, estimaciones, y reparto de tareas), para comenzar a desarrollar.

Las características del Sprint Backlog son:

- Las estimaciones de tiempo tendrán una duración de entre 4 y 16 horas. En caso de valores superiores, se dividirán.
- Las tareas deben ser estar enfocadas a la realización de una de las necesidades del Product Backlog.
- El avance del progreso y la velocidad se controlarán mediante gráficos ***Burndown Chart*** (explicado más adelante).

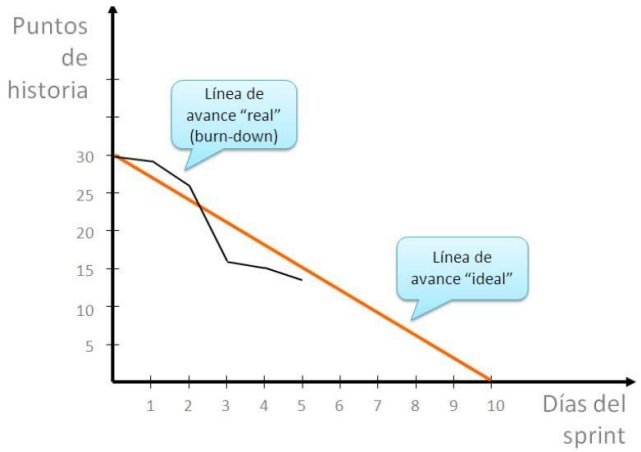

Gráfico de Burndown de seguimiento.

El sprint backlog puede presentarse y, por tanto, ser controlado y actualizado en distintos soportes:

- Hojas de cálculo (tipo Excel)

- Pizarra.
- Soportes digitales (tipo Jyra por ej.).

La herramienta más usada es el ***Scrum Taskboard*** consistente en una pizarra, en la que se reflejan las siguientes entidades:

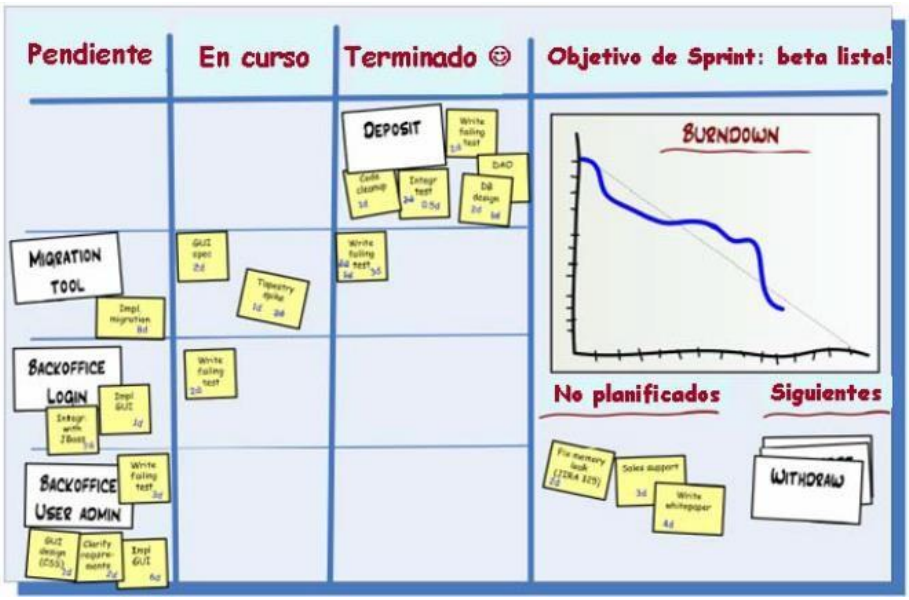

Figura 29. Tablero Scrum Taskboard. (Kniberg et al. 2007)

- **Tareas Pendientes**: Tareas planificadas que aún no se han comenzado.
- **Tareas en curso**: Tareas planificadas que están siendo llevadas a cabo, por una o varias personas del equipo.
- **Tareas Terminadas**: Tareas planificadas finalizadas.
- **Tareas no planificadas**: Tareas no planificadas pero que puede ser interesante tenerlas como referencia.
- **Tareas Siguientes**: Tareas no planificadas, que pueden ser llevadas a cabo si se acaban todas las tareas del Sprint antes de que este finalice.
- **BurnDown**: Gráfico de avance, que irá reflejado el avance diario.
- **Impedimentos**: También suele reflejarse una zona donde se colocan aquellas tareas bloqueantes.
- **Retrospectiva**: Otra zona donde se reflejan las enseñanzas que merezca tener presentes, tanto positivas como negativas.

4.2.5.4 Estimación del Sprint: Planning Póker

La estimación de la complejidad de las distintas tareas que pueden componer un sprint es una tarea de análisis que debe ser consensuada con todo el equipo que conforma el squad.

Para poder llevarla a cabo, de una forma desenfadada, y que las distintas opiniones no condiciones al resto de participantes, se recurren a técnicas de opinión indirectas, como la del **Planning Póker.**

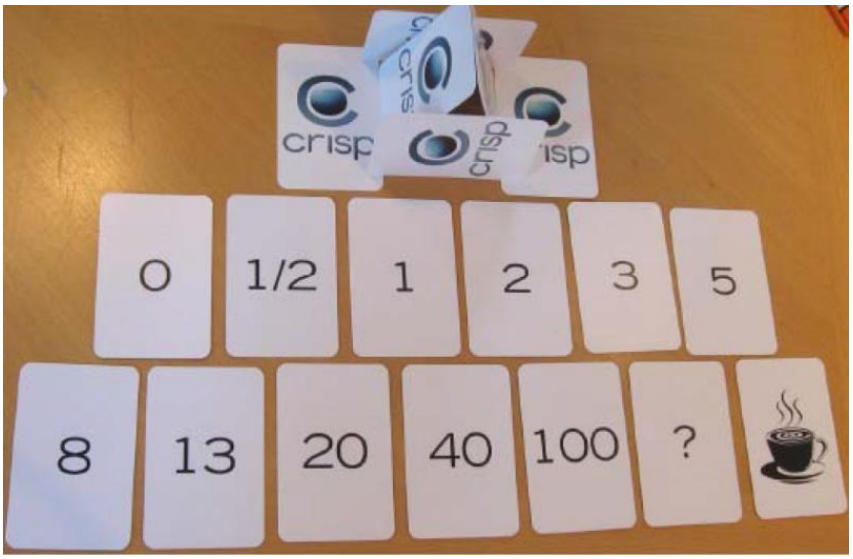

Figura 30.Baraja de cartas de Planning Póker.

El juego de valoración se lleva de la siguiente manera:

- Se reparte a cada miembro una baraja de 13 cartas.
- Se muestra una historia de usuario y cada miembro ha de elegir una carta en función de su estimación de dificultad/esfuerzo, y ponerla boca abajo.
- Cuando todos han escogido carta, se les da la vuelta al mismo tiempo.
- Se revisan las cartas, y si existiesen grandes diferencias, se debate (siendo los valores extremos quienes tengan que justificar su valoración) y se tratan las razones para haber valorado de dicha forma la historia. Se repite posteriormente la votación, hasta que las estimaciones sean parejas o uniformes.
- El valor estimado es el valor de puntos de trabajo necesarios para el desarrollo de dicha historia, de ahí que la secuencia de números no sea lineal y, por ejemplo, haya un salto entre 40 y 100. De esta forma se evitan sensaciones falsas, para estimaciones grandes.

La carta 0 se utiliza cuando una historia ya está hecha o requiere muy poco tiempo de trabajo.

La carta ¿? Se emplea cuando se puede llegar a tener una idea de la complejidad de la tarea.

La carta con la taza de café se emplea para pedir un descanso después de la estimación.

4.2.5.5 Actualización del diagrama de Burndown

Es importante que el diagrama de Burndown esté actualizado para poder tener feedback del avance de los sprint, y poder tomar decisiones de forma rápida. Para ello planteamos varias opciones que podrían darse en el desarrollo de un sprint:

En este ejemplo 1, se estima un ritmo de trabajo constante con pendiente negativa, y en la realidad, a día 16, se observa que, según el ritmo que se desarrolla, se completaría el Sprint antes del día 19.

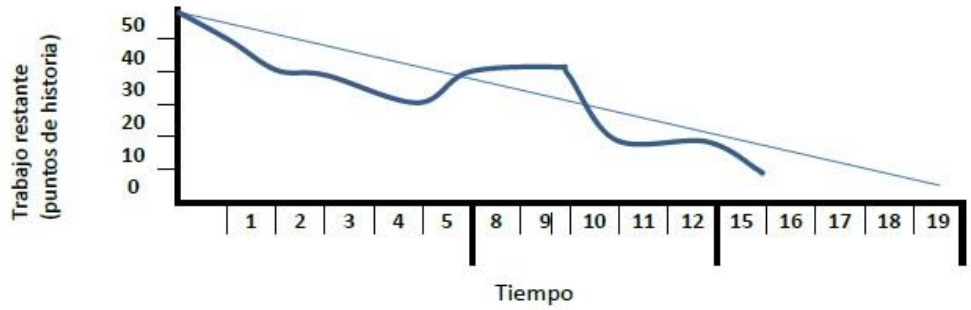

Figura 31. Ejemplo 1 de diagrama de burndown No se tienen en cuenta los días no laborables para no alterar los diagramas.

En este 2º ejemplo, los puntos de historia se han realizado antes de lo previsto, pudiendo realizar más puntos de historia antes de que acabe el sprint, y así adelantar tareas.

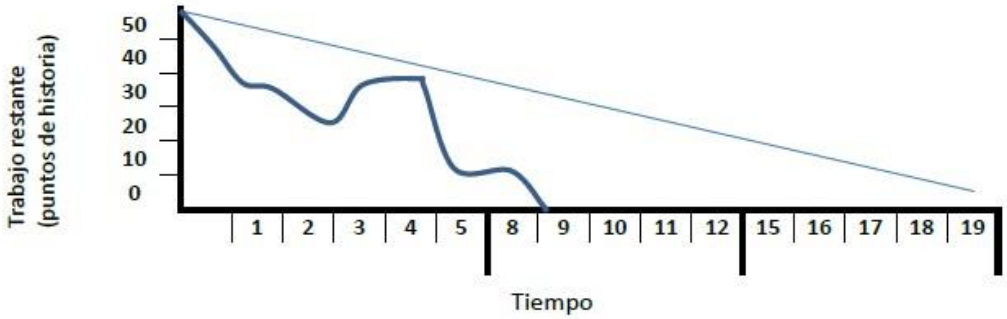

Figura 32. Ejemplo 2 de diagrama de burndown

En este último caso, el diagrama refleja una carga excesiva de tareas a realizar, siendo necesario analizar qué tareas habría que sacar de dicho sprint, de cara a planificarlos para el siguiente.

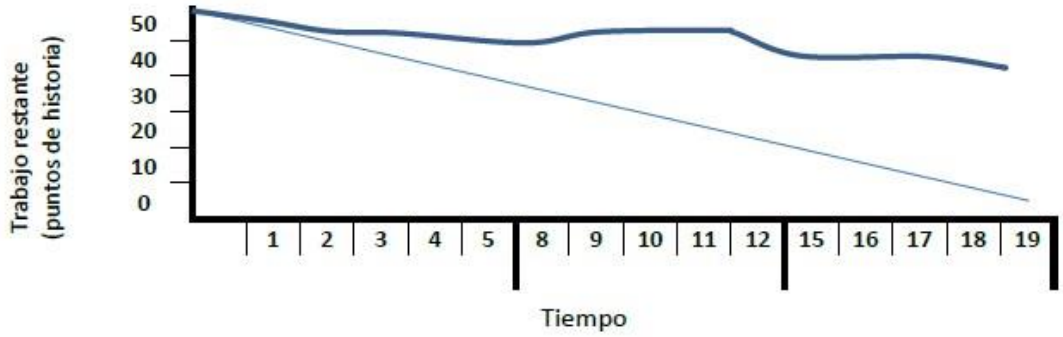

Figura 33. Ejemplo 3 de diagrama de burndown

Como recomendación general, el intervalo adecuado para los sprints está entre las 2 y las 4 semanas.

4.2.5.6 Reuniones durante el sprint, y post-sprint

Durante el desarrollo de los sprints, y al finalizar el sprint, hay que destacar 2 tipos de reuniones necesarias y que aportan un gran valor al proceso:

1. **Reunión Diaria** (*Sprint Daily Meeting*)

En esta reunión diaria se busca obtener un estado de situación de cada miembro de cara a mejorar el rendimiento de cada uno.

Teniendo como referencia el Sprint Backlog, y el gráfico de Burndown con la información del día anterior, y con las tareas que tiene asignado cada uno, en esta reunión, que debe durar menos de 15 minutos y que se hace de pie, se contestan a 3 preguntas básicas:

- **¿Qué hiciste ayer?**
- **¿Qué vas a hacer hoy?**
- **¿Qué problemas has tenido para realizar tus tareas?**

Con esta reunión, se busca el compromiso de todo el equipo, evitando ausencias o bajo ritmo de trabajo.

2. **Reunión** de **Revisión** del Sprint (*Sprint Review Meeting*)

En esta reunión, el Scrum Master, junto con el equipo, presentan los productos entregables que se hayan realizado de forma que tanto usuarios, como Product Owner, u otros actores involucrados, puedan analizar el producto entregado y los problemas que hayan podido encontrar durante el sprint.

Las características de esta reunión son:

- Duración máxima de 4 horas.
- Se presenta el producto "entregado", de acuerdo con las definiciones que se hicieron como "entregables".
- Si la funcionalidad no está completa no se presenta.
- La funcionalidad se presenta en un entorno lo más parecido al de producción.

En esta reunión se comprueba si se ha entendido lo que necesitaba el usuario, y si se cubren todos los requisitos.

3. **Reunión** de **Retrospectiva** (Sprint Retrospective Meeting)

En esta reunión, el equipo analizará el sprint que ha terminado y sacará recomendaciones a aplicar en próximos sprints.

Esta reunión está convocada por el Scrum Master, y suele durar unas 3 horas.

Las características de la reunión son:

- Asisten Scrum Master, equipo y el Product Owner.
- Los temas de la reunión serán: ¿Qué ha ido bien en el último Sprint?, ¿Qué podemos mejorar para el siguiente Sprint?
- El Scrum Master tomará notas de las respuestas.
- El equipo comentará las posibles mejoras que se puedan realizar, indicando cuáles van a tener preferencia.
- El Scrum Master tratará de ayudar a que estas mejoras se implementen para el siguiente sprint.

4.2.6 Resumen

La metodología Scrum, se centra en la gestión del proyecto incorporando la gestión de cambios como una tarea más en el día a día y puede aplicarse a todo tipo de proyectos.

Busca el trabajo cooperativo de equipos multidisciplinares y de alto rendimiento. Para ello, se definen sprints relativamente cortos (2-3 semanas) de duración fija y se determinan una serie de reuniones a mantener a lo largo de todo el proyecto, ya sean antes, durante, o después de los sprints.

Figura 34. Reunión de Scrum.

Como resultado de cada sprint, se obtienen incrementos y, debido a la realización de un control de proceso continuo, es adaptable a nuevas necesidades o cambios del negocio, de una forma ágil y rápida, siempre respetando las normas mínimas acordadas antes de comenzar el sprint, como son el compromiso de no interrumpir el proceso de elaboración de cada sprint, o cambiar las historias de usuario dentro del sprint.

Es necesario un compromiso entre el **Product Owner**, dueño del producto, que toma decisiones y prioriza las historias de usuario, y el **Scrum Master**, que es la cabeza visible del equipo de trabajo, desarrollador de cada una de las tareas de usuario que compondrán cada historia.

Scrum plantea un trabajo en equipo, consensuado, y en el que todas las partes del proyecto apunten hacia el mismo lugar en todo momento: la participación del usuario en el desarrollo de los productos en cada sprint, hace que éste se sienta un miembro más del equipo, pasando de frases del estilo: “habéis construido…”, “habéis hecho”, al “hemos construido” o “hemos hecho”.

Ese cambio de percepción tan sutil introduce varios aspectos que, hasta ahora en la gestión de proyectos, no se tenían en cuenta, como son conciencia y colaboración.

Mediante el desarrollo ágil, el usuario es consciente de que las cosas que pide, las funcionalidades que reclama tienen un esfuerzo en tiempo y dinero, y que los errores existen.

5 Comparativa de metodologías

5.1. Introducción

Este capítulo trata sobre la comparación de las metodologías descritas en el capítulo 3. Como ya se ha dicho, se llama a todas metodologías cuando algunas de ellas (sobre todo las ágiles) son más bien un conjunto de principios, valores y buenas prácticas. Sin embargo, la comparación sirve para explicar cómo abordan las distintas áreas de las que se componen la gestión de proyectos.

Hay una primera sección que se compara las metodologías tradicionales y las ágiles desde un punto de vista general, sin entrar en concreciones propias de cada metodología.
A continuación, se muestra una tabla comparativa entre dos representantes principales de cada una de las dos familias. En este caso, se ha elegido PMBOK y Scrum por estar más extendidas y por ser las que el autor conoce por su experiencia profesional.

Por último, en la siguiente sección se aborda la cuestión de cómo elegir la metodología más apropiada dependiendo de las características del proyecto.

5.2. Comparación entre las metodologías tradicionales y ágiles

A diferencia de las metodologías tradicionales, las metodologías ágiles surgen como una variante para dar respuesta a determinados problemas. Como bien indica el artículo "*Hallazgos empíricos en Métodos Ágiles*" (Lindvall et al. 2002) las metodologías ágiles son adecuadas cuando los requisitos son emergentes, y cambian de forma rápida.

A todo esto, se suma que en las metodologías tradicionales los **riesgos** se acumulan, siendo críticos si surgen en la etapa final, repercutiendo en retrasos, o incumpliendo la ejecución de las últimas fases del ciclo del proyecto.

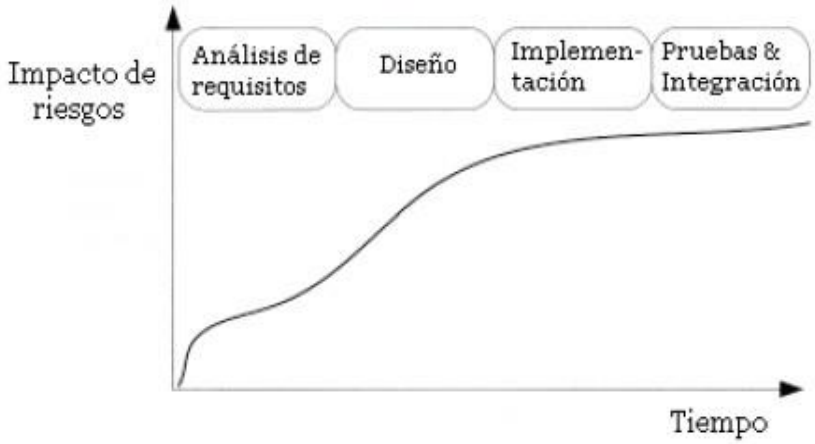

Figura 35. Impacto de riesgos vs Evolución de un proyecto Waterfall

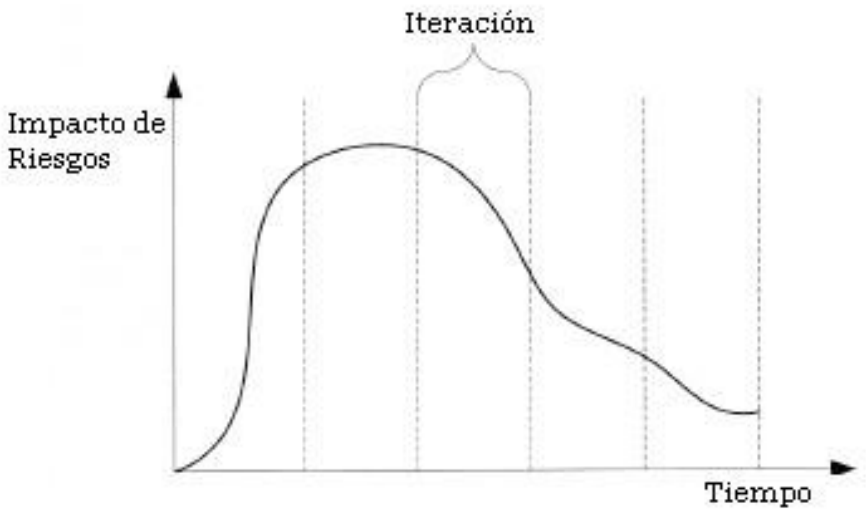

Figura 36. Figura 6. Impacto de riesgos vs Evolución de un proyecto Ágil

De este modo, las metodologías ágiles presentan varias ventajas:

1. **Respuesta a los cambios durante el desarrollo** puesto que durante toda la vida del desarrollo existe la posibilidad de mejorar el producto, y por tanto la percepción del cliente hacia el mismo, ya que es partícipe, y además, se integra la gestión de cambios como una característica del proceso.
2. **Entrega continúa en plazos breves** permitiendo al cliente saber en todo momento el estado de avance del proyecto, pudiendo probar las funcionalidades del producto incrementalmente y comprobar si cumplen sus necesidades, mejorando su satisfacción. Además, el trabajo y evaluación en periodos cortos de tiempo, disminuye el tamaño de los riesgos, las dificultades a dichos ciclos, y permite ir aprendiendo de los mismos.
3. **Trabajo involucrando al cliente con equipo de desarrollo** con una comunicación directa y fluida eliminando malentendidos (principal origen de errores), y documentación que no aporta valor.
4. **Importancia de la simplicidad** eliminando trabajos innecesarios.
5. **Gestión de cambios integrada** en el propio proceso.
6. **Mejora continua de los procesos** y del equipo de desarrollo, dependientes del éxito técnico, del éxito personal y del éxito organizacional.

En la siguiente tabla, podemos hacer una comparación general entre las metodologías tradicionales y las metodologías ágiles:

TRADICIONALES	AGILES
Orientada a proyectos de cualquier tamaño	Orientada a proyectos pequeños
Equipos grandes y dispersos	Equipos pequeños, sobre 10 personas
Proyectos de media / larga duración	Proyectos de corta duración
Proyecto cerrado	Proyecto abierto a cambios
El cliente mantiene reuniones con la dirección	El cliente está integrado en el equipo
Arquitectura prefijada	Arquitectura se va mejorando
Documentación rigurosa	Poca documentación
Roles específicos	Roles genéricos
Roles no intercambiables	Roles flexibles
Centrada en los procesos	Centrada en las personas
Gestión dirigida	Gestión colaborativa
Alto coste de prototipado	Bajo coste de prototipado
Planificación inicial alta	Planificación inicial baja
Basada en estándares de desarrollo	Basadas en heurísticas
Poco feedback	Continuo feedback
Proceso lineal	Proceso iterativo
El coste se acerca a lo estimado	El coste puede dispararse

Tabla 1. Comparación general entre Metodologías Tradicionales y Ágiles

Ahora pasamos a comparar dos metodologías representativas de cada una de las diferentes familias. Por el lado de las metodologías tradicionales, se ha elegido la PMBOK como una de las más representativas.

Por el lado de las metodologías ágiles, se ha elegido Scrum, de la que se ha hablado extensamente en este trabajo.

Ambas metodologías forman parte de la gestión de proyectos que el autor conoce desde su experiencia profesional. Por eso resulta interesante comparar ambas ya que resulta a veces complicado entender que enfoque es más apropiado para los diferentes proyectos IT a los que nos podemos enfrentar.

ÁREAS	PMBOK	SCRUM
Integración	• Desarrollar acta constitución del proyecto • Desarrollar el plan del proyecto • Dirigir la ejecución del proyecto • Monitorizar y controlar el trabajo del proyecto • Realizar control integrado de cambios • Cerrar el proyecto	• Verificar aprobación proyecto y su financiación en la fase de planificación • Validar herramientas e infraestructura en la fase de planificación • Fuerte gestión de cambios con el producto y sprint backlog • Refinamiento arquitectura sistema para apoyar los cambios
Involucrados	• Identificar involucrados • Plan de involucrados • Dirigir relación involucrados • Controlar relación involucrados	
Alcance	• Plan de alcance • Recopilar requisitos • Definir alcance • Crear WBS • Verificar alcance • Controlar alcance	• Analizar y construir modelo esquemático del proyecto • Hacer Backlog del proyecto • Definir el sprint • Definir funcionalidades incluidas • Selección versión más adecuada desarrollo inmediato • Revisar progreso tareas asignadas
Plazos	• Plan de plazos • Definir actividades • Secuenciar actividades • Estimar recursos de actividad • Estimar duraciones de actividad • Desarrollar calendario • Controlar calendario	• Fijar fechas y funcionalidades para cada versión • Iteraciones mensuales
Coste	• Plan de costes • Estimar costes Determinar el presupuesto • Controlar los costes	• Estimación del coste de la versión en la fase de planificación
Calidad	• Plan de calidad Realizar aseguramiento de la calidad • Realizar control de calidad	• Revisión y ajuste de las normas con el que el proyecto se acordó Reunión revisión del diseño • Reunión planificación de sprint • Reunión revisión de sprint. • Scrum diario
Recursos humanos	• Plan de RRHH • Adquirir el equipo • Desarrollar el equipo • Dirigir el equipo	• Nombramiento del equipo de proyecto para cada versión • Participación del equipo en reuniones de sprint • Participación del equipo en scrums diarios
Comunicaciones	• Plan de comunicaciones • Dirigir las comunicaciones • Controlar las comunicaciones	• Comunicación de las nomas del proyecto al equipo • Reunión revisión del diseño • Reunión planificación de sprint • Scrum diario
Riesgos	• Plan de riesgos • Identificar riesgos • Análisis cualitativo del riesgo • Plan de respuesta • Monitorizar y controlar los riesgos	• Evaluación del riesgo al inicio del proyecto • Revisar los riesgos en las reuniones de revisión (retro/review)
Aprovisionamiento	• Plan de aprovisionamientos • Realizar aprovisionamiento • Administrar aprovisionamiento • Cerrar aprovisionamiento	

Tabla 2. Comparación metodologías PMBOK vs Scrum

5.3 ¿Agile o Waterfall?

Aunque el objetivo de este TFM no estudiar un método para la decisión de la metodología más apropiada para cada proyecto, si parece interesante presentar algunas cuestiones importantes que se pueden dar a la hora de decidirse por un tipo de metodología u otra.

Independientemente de que metodología se use, todas siempre tienen que cubrir un cierto nivel de administración y deberán adaptarse lo mejor posible a las características del proyecto.

Las siguientes consideraciones pueden ayudar a elegir entre un enfoque ágil o un enfoque tradicional.

5.3.1 Contexto tecnológico

Este es un factor principal en el entorno de la Industria 4.0. Dependiendo del nivel de madurez de la tecnología que se piensa utilizar en el proyecto, podemos estimar qué metodología nos conviene utilizar.

- ¿Hay un plan claramente definido?
- ¿Como de estable es el alcance del proyecto?
- ¿Cuál es el grado de definición de los requisitos del proyecto?

Usando una matriz de Stacey podemos identificar más fácilmente qué enfoque es más adecuado

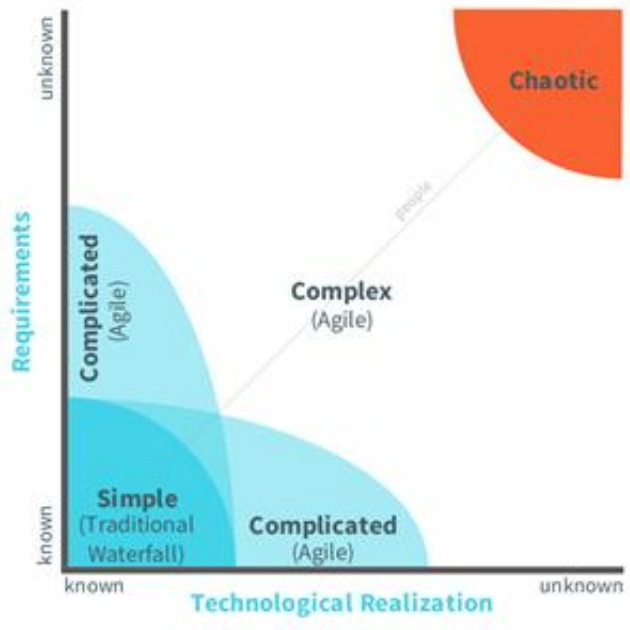

Figura 37. Matriz complejidad requerimientos tecnología

5.3.2 Relación con los principales stakeholders

Se debería tener claro cuál va a ser el papel de los principales involucrados en el proyecto y cuál va a ser su relación y grado de colaboración.

Un proveedor externo, una consultora o cualquier otro involucrado relevante en el proyecto debe mostrar un compromiso de participación y comunicación importante si lo que queremos es elegir una metodología agile.

En este caso tendríamos que hacer un esfuerzo por definir claramente los roles y funciones tanto en el lado del gestor como en de los involucrados.

Por el contrario, si su relación con el proyecto se va a limitar con entregar un producto en forma y fecha determinados y no muestra disponibilidad para colaborar de otra manera o bien no tiene capacidad para ello, quizás debamos pensar en una metodología más tradicional.

En este caso prestaremos atención a la documentación, a las fechas de entrega, al cumplimiento de plazos.

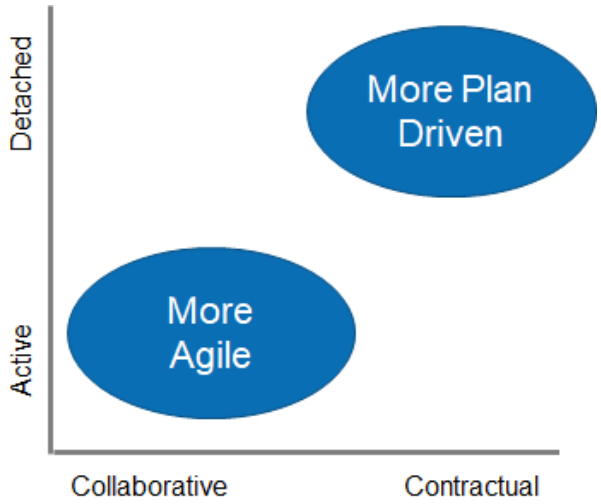

Figura 38. Matriz relación con stakeholders

5.3.3 Nivel de conocimiento del equipo del proyecto

Un enfoque ágil normalmente demanda un novel de formación y sofisticación del equipo mayor. Se deben considerar las habilidades y debilidades del equipo.

Incluso en un contexto favorable para una metodología ágil (por complejidad, requerimientos etc…) y con unos proveedores/involucrados dispuestos a colaborar, aun se puede fracasar debido a que el equipo no está debidamente preparado y formado para trabar con dicha metodología.

5.3.4 Tamaño del proyecto

La experiencia ha demostrado que la relativamente baja tasa de éxito está relacionado con plazos de entrega inalcanzables o presupuestos demasiados optimistas sin utilizar métricas para llegar a estimaciones aproximadas. Las empresas maduras en desarrollo de software son atinadas a la hora de estimar porque, a lo largo de su vida como empresa, han desarrollado métricas y herramientas eficientes para medir los costes y plazos.

Esto nos lleva a pensar también que gestionar proyectos pequeños es en la práctica más fácil que estimar grandes proyectos. Esta intuición de sentido común queda patente en la figura 39

con unos porcentajes de éxito muy distinto según sea la envergadura. Luego sigue siendo un reto a día de hoy gestionar grandes proyectos y parece lógico pensar que no es por falta de medios ya que estos grandes proyectos los realizan grandes multinacionales de IT.

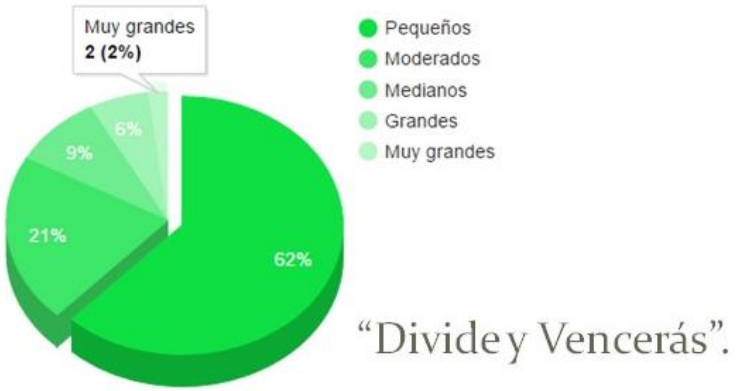

Figura 39. Estado de los proyectos a su finalización según su tamaño (pequeños y grandes). Dato de CHAOS Manifesto 2015

Otro punto de vista interesante a la hora de analizar el éxito de los proyectos es clasificarlos por las metodologías con las que fueron realizados. De esta forma se pueden extraer conclusiones si unas metodologías alcanzan mayores tasas de éxito que otras.

En la figura 40 se dividen proyectos pequeños según metodologías ágiles y tradicionales, siendo los porcentajes de éxito, cuestionados y fracasados bastante similares. Por tanto, no podemos decir a priori que un tipo de metodologías sean más exitosas que otras.

Sin ánimo de ser exhaustivo, se puede concluir diciendo que los proyectos software se enfrentan a muchos y diversos factores que dificultan finalizarlos correctamente. La causa principal es que el software es el producto de una actividad intelectual realizado por seres humanos, con todo lo que eso supone.

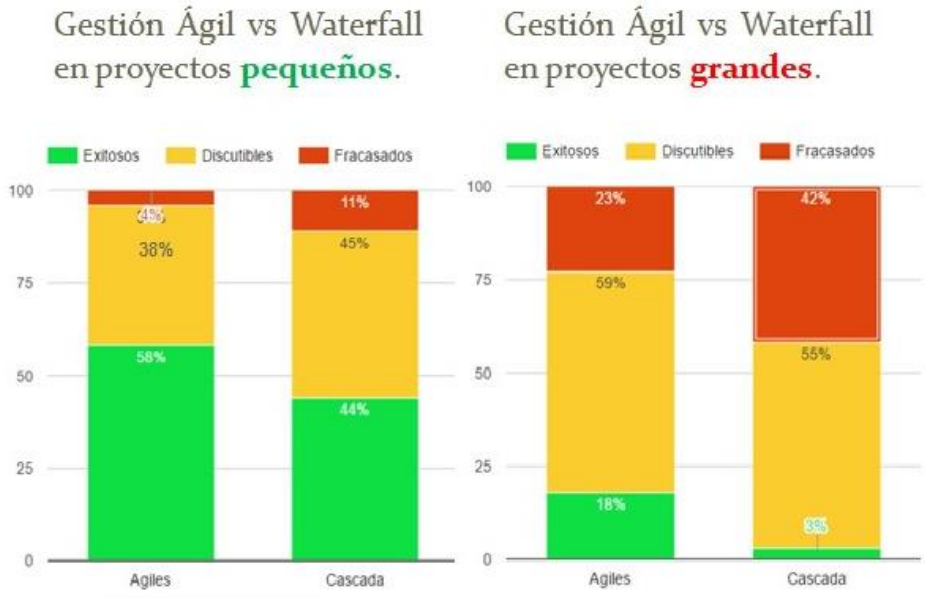

Figura 40. Estado de pequeños proyectos a su finalización realizados con metodologías ágiles o tradicionales (waterfall). Dato de CHAOS Manifesto 2015.

Todos estos factores de decisión podríamos agruparlos en una tabla como esta, en la que podamos dar puntuación a cada uno de los factores y esto nos ayude a decidir qué enfoque podría ser más adecuado para cada proyecto.

5.3.5 Resumen

A modo de resumen y como ayuda para contestar a las anteriores cuestiones y a algunos otros factores importantes, podríamos utilizar el siguiente formulario y se rellenarlo según las necesidades del proyecto, (solo se puede marcar una opción por factor):

FACTOR	VALOR			
Prioridad de negocio	Cumplimiento - Valor			
	○	○	○	○
Tiempo del proyecto	Largo - Corto			
	○	○	○	○
Tamaño del equipo	Grande - Pequeño			
	○	○	○	○
Interacción con el cliente	Mínima - Máxima			
	○	○	○	○
Ambiente del desarrollo	Controlado - Incertidumbre			
	○	○	○	○
Dirección	Organizativa - Colaborativa			
	○	○	○	○
Rigidez del producto	Cerrado - Ampliable			
	○	○	○	○
Requisitos	Claros - Ambiguos			
	○	○	○	○
Riesgo de fallos	Bajos - Altos			
	○	○	○	○
Enfoque de desarrollo	Procesos - Personas			
	○	○	○	○
Necesidad de documentación	Alta - Baja			
	○	○	○	○
Probabilidad de cambios en el equipo	Alta - Baja			
	○	○	○	○
Roles intercambiables	No flexible - Flexible			
	○	○	○	○
Frecuencia de entregables	Baja - Alta			
	○	○	○	○
VALORES	**1**	**2**	**3**	**4**
TOTAL POR COLUMNAS	**A**	**B**	**C**	**D**
VALORACIÓN FINAL	**(A+B+C+D) / 14 = VALORACIÓN**			

Figura 41. Formulario orientativo elección metodología

Se rellena, se suman los valores y se divide entre el número de factores.

Si el resultado está más cerca del 1, sería más óptima una metodología tradicional, y si es más cercano a 4, sería más recomendable una metodología ágil.
Los valores corresponden a:

1. Acorde total con enfoque tradicional
2. Acorde parcial con enfoque tradicional
3. Acorde parcial con enfoque ágil
4. Acorde total con enfoque ágil

De una forma visual se colocaría el resultado en la siguiente escala:

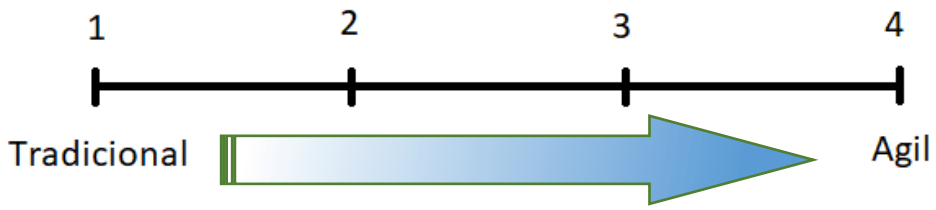

Figura 42. Escala enfoque tradicional vs ágil

6 Conclusiones finales

6.1. Conclusiones

Este TFM ha pretendido realizar una presentación general de la los proyectos relacionados con las tecnologías de la información, en concreto en el mundo de la industria moderna en la etapa actual. A esta renovación y revisión de la industria y sus procesos se les ha bautizado como "cuarta revolución industrial" o directamente "Industria 4.0"

Las metodologías de gestión de proyectos están en continua evolución, más aún cuando hablamos de desarrollo de software. Se ha pretendido hacer una introducción y descripción de aquellas que parecen estar más extendidas en detalle, tal y como el autor las conoce por su propia experiencia.

Finalmente se ha hecho una breve comparación de las dos familias de metodologías. Además, se intentado hacer una serie de recomendaciones a la hora de decidir cuál es la metodología más adecuada para un tipo de proyecto atendiendo a diferentes factores y cuestiones.

Algunas conclusiones u observaciones relevantes, a juicio del autor, que han quedado patentes mientras se realizaba el TFM.

- Los proyectos con un alto contenido de software y tecnología como lo que se han descrito aquí, parecen ser buenos candidatos para aplicar algunas de las técnicas ágiles en su gestión.
- He intentado reflejar que no existe una metodología única y perfecta que garantice el éxito a cualquier tipo de proyecto.
- Se ha simplificado llamando metodologías a todas las formas de gestionar proyectos aquí tratadas cuando algunas de ellas no son metodologías propiamente dichas, sino más bien guías de buenas prácticas o directrices para gestionar proyectos.
- La clave a la hora de gestionar y dirigir un proyecto es saber cuál es la metodología más recomendable. Para cada proyecto hay que encontrar un equilibrio entre agilidad y la planificación tradicional.
- Se puede utilizar una metodología que se ajuste a un proyecto como base y a partir de ahí, modificarla adaptándola a las necesidades del proyecto.

En algunas grandes empresas existen ya guías o recomendaciones para la gestión de sus proyectos donde se puede apreciar la aplicación de diferentes partes de metodologías ágiles y tradicionales, buscando sacar lo mejor de cada una de ellas.

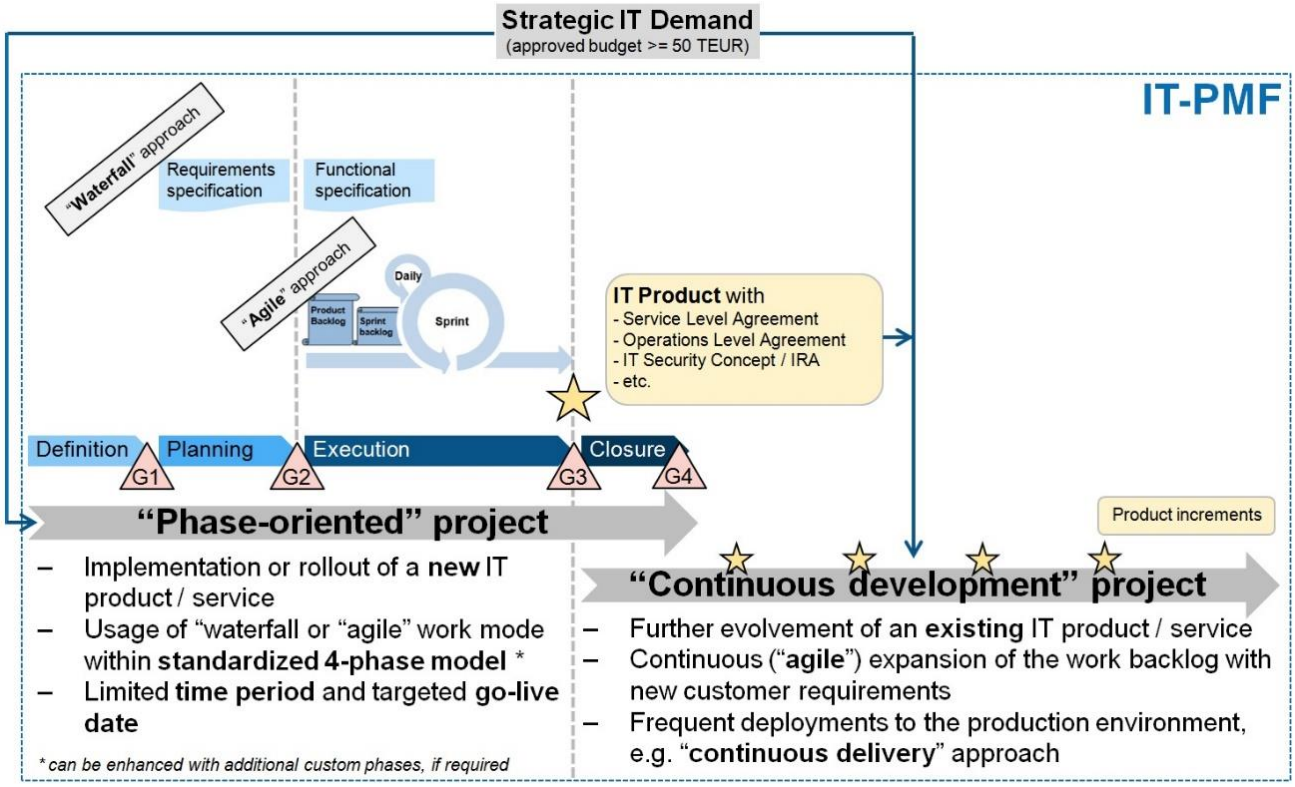

Figura 43. Ejemplo de guía de proyectos IT de una empresa

Este es un ejemplo de la oficina de gestión de proyectos de una empresa y se puede ver como recomiendan utilizar un enfoque "waterfall" para las fases de definición y planificación del proyecto.

Sin embargo, para la fase de Ejecución recomiendan un enfoque ágil, basado en Scrum. Al igual que para el ciclo de vida posterior del producto, para su mantenimiento y mejora, recomiendan un enfoque agile.

El uso del método ágil aplicado al desarrollo de proyectos trae consigo una serie de **ventajas**, experimentadas a lo largo de su uso como pudieran ser:

- Acelera la entrega de productos digitales.
- Gestionar prioridades cambiantes.
- Aumenta la eficacia de los equipos.
- Alineamiento entre Negocio y Tecnologías de la Información (TI).
- Mejora el *time-to-market*.
- Aumenta la satisfacción del cliente.
- Captación y retención de talento.
- Entorno colaborativo y de innovación.
- Entrega constante de valor.
- Equipos empoderados y autoorganizados.
- Feedback constante del cliente.

Sin embargo, a pesar de traer todas estas mejoras, existen otros factores en los cuales se podría incidir, para tratar de mejorar:

- **Síndrome del Burn Out:** trabajar con métodos ágiles puede aumentar el estrés de algunos de los desarrolladores.
- **Falta de participación del cliente:** a pesar del entusiasmo por trabajar en base a métodos ágiles, a la hora de la verdad, el cliente no lo pone tan fácil para mantener el ritmo de comunicación necesario.
- **Aumento del riesgo:** el ritmo del trabajo y el entusiasmo por los logros alcanzados puede hacer perder la aversión al riesgo que se necesita en cualquier proyecto.
- **Falta de controles de calidad:** la calidad no puede obviarse. De nada sirve avanzar a toda velocidad, si el producto no reúne las condiciones necesarias.
- **Falta de documentación:** Las metodologías ágiles apuestan por un software que funciona por etapas, dejando en un segundo plano la gestión documental, porque supuestamente ésta se intregra como control de cambios en el proceso.
- **Fuerte dependencia de los líderes:** Los equipos de trabajo dependen del liderazgo de la persona responsable del proyecto. Las continuas reuniones y evaluaciones periódicas hacen que la persona responsable del proyecto centralice gran parte de las responsabilidades y decisiones.

6.2 Próximas áreas de desarrollo.

Se proponen como futuras áreas de desarrollo en proyectos relacionados con las metodologías ágiles, los siguientes temas:

- Comparativas de estudio *waterfall- agile* para un mismo proyecto.
- Escalabilidad de *Agile* en otros niveles dentro de una organización empresarial. Por ejemplo SAFe (Scaled Agile Framework)
- Metodologías híbridas como una adaptación de ambos enfoques
- Lean como resultado de la aplicación de agile.
- Uso de metodologías agiles dentro y fuera de España.
- Uso de Watergile, o Agilefall.

La propuesta es continuar en el estudio y análisis de los enfoques ágiles para la gestión de proyectos no puramente de desarrollo de software. Parece que algunas de sus ideas pueden ser aplicables a otro tipo de proyectos, como por ejemplo los IT.

En los proyectos de desarrollo de software parece que las ventajas parecen estar claras y eso ha ayudado a que se extiendan por todo el sector. Lo que no he podido contrastar durante este trabajo es si realmente su aplicación a proyectos de otra índole tiene más ventajas que inconvenientes.

7 REFERENCIAS

1. *A guide to the project management body of knowledge (PMBOK guide), 6th edition*. Project Management Institute, 2013. ISBN-13: 978-1628251845
2. *Software Extension to the PMBOK Guide*. Project Management Institute, 2013. ISBN-13: 978-1-6282-5013-8.
3. *IPMA Competence Baseline (ICB), version 3.0*. International Project Management Association, 2006. ISBN: 0-9553213-0-1.
4. *Managing Succesful Projects with PRINCE2*. The Stationery Office, 2009. ISBN: 978-0-11-331059-3
5. P. Bourque and R.E. Fairley, eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK), version 3.0*. IEEE Computer Society, 2014. ISBN-13: 9780-7695-5166-1
6. *UNE-ISO 21500, Directrices para la dirección y gestión de proyectos*. AENOR, 2013.
7. Ken Schwaber and Jeff Sutherland, *La Guía Definitiva de Scrum: Las Reglas del Juego*. 2013.
8. ALLAN KELLY. *Changing software development: learning to be agile*. John Wiley & Sons Ltd, 2008. ISBN-13: 978-0-470-51504-4.
9. DEAN LEFFINGWELL. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley, 2011. ISBN-13: 978-0-32163584-6.
10. JAMES SHORE and SHANE WARDEN. *The art of agile development*. O'Reilly, 2008. ISBN-13: 978-0-596-52767-9.
11. MIKE COHN. *Agile estimating and planning*. Prentice Hall, 2005. ISBN-13: 978-0131-47941-8.
12. ROGER S. PRESSMAN. *Software engineering : a practitioner's approach, 7th edition*. Mc Graw Hill, 2010. ISBN: 978-0-07-337597-7.
13. IAN SOMMERVILLE. *Software Engineering, 9th edition*. Addison-Wesley, 2011. ISBN-13: 978-0-13-703515-1.
14. POLLYANNA PIXTON, PAUL GIBSON and NIEL NICKOLAISEN. *The agile culture*. Addison-Wesley, 2014. ISBN-13: 978-0-321-94014-8.
15. Constantin Scheuermann, Stephan Verclas, Bernd Bruegge: *Agile Factory - An Example of an Industry 4.0 Manufacturing Process.* 2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications 2015
16. Miloš Jovanović, Bojan Lalić, Antonia Mas, Antoni-Lluís Mesquida. *THE AGILE APPROACH IN INDUSTRIAL AND SOFTWARE ENGINEERING PROJECT MANAGEMENT.* Paper number: 13(2015)4, 331, 213 - 216

17. THE STANDISH GROUP. *CHAOS Manifesto 2015*. 2015. http://www. standishgroup.com
18. Aitken, Ashley; Ilango, Vishnu, *A Comparative Analysis of Traditional Software Engineering and Agile Software Development*. System Sciences (HICSS), 2013 46th Hawaii International Conference.
19. Larman, C.; Basili, V.R., *Iterative and incremental developments. a brief history*. Computer, vol.36, no.6, pp.47,56, June 2003.
20. Abrahamsson, P.; Warsta, J.; Siponen, M.T.; Ronkainen, J., *New directions on agile methods: a comparative analysis*. Software Engineering, 2003. Proceedings. 25th International Conference.
21. Coram, M.; Bohner, S., *The impact of agile methods on software project management*. Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops.
22. Sulaiman, T.; Barton, B.; Blackburn, T., *AgileEVM - earned value management in Scrum Projects*. Agile Conference, 2006.
23. Fitsilis, P., *Measuring the Complexity of Software Projects*, Computer Science and Information Engineering, 2009 WRI World Congress on , vol.7, no., pp.644,648, 2009.
24. J. Laurenz Eveleens and Chris Verhoef, *The rise and fall of the Chaos report figures*.
25. IEEE Computer Society, 2010.
26. B. M. Moniruzzaman and Syed Akhter Hossain, Comparative Study on Agile software development methodologies. Journal CoRR, volume abs/1307.3356, 2013.
27. Fitsilis, P., Comparing PMBOK and Agile Project Management software development processes. Computer Science and Information Engineering, 2009 WRI World Congress on, vol.7, no., pp.644,648, 2009.

Trabajos académicos:

1. Pilar Rodríguez González, Estudio de la aplicación de metodologías ágiles para la evolución de productos software. Trabajo Fin de Máster, Facultad de Informática, Universidad Politécnica de Madrid, Septiembre 2008.
2. Pedro Sáez Martínez, Identificación y valoración de técnicas ágiles de gestión de proyectos software. Trabajo Fin de Máster en Dirección de Proyectos, Universidad de Oviedo, Julio 2013.
3. Luis Álvarez Puertas, Oficina de Gestión de Proyectos Ágil: control y seguimiento de proyectos ágiles. Trabajo Fin de Máster en Dirección de Proyectos, Universidad de Oviedo, Junio 2013.
4. Jesús Santiago Rial Huerta. Aplicación de Metodologías Ágiles a Desarrollo de Proyectos. Trabajo Fin de Máster en Dirección de Proyectos, Universidad de Sevilla, 2019.

Páginas web consultadas:

1. Universidad politécnica internacional: http://www.upi.ac.cr/blog/2018/03/06/relacion-de-metodologias-agiles-con-la-industria-de-manufactura

2. Instituto tecnológico de Buenos Aires: https://www.itba.edu.ar/la-universidad/prensa/metodologias-agiles-un-cambio-de-chip/
3. Página de International Organization for Standardization (ISO): http://www.iso.org
4. Página de Institute of Electrical and Electronics Engineers (IEEE): http://www.ieee.org
5. Página de la Institute of Electrical and Electronics Engineers (IEEE) Computer Society: http://www.computer.org
6. Página de Asociación Española de Normalización y Certificación (AENOR): https://www.aenor.es
7. Página de Software Engineering Institute (SEI): http://www.sei.cmu.edu
8. Página de Project Management Institute (PMI): http://www.pmi.org
9. Página de International Project Management Association (IPMA): http://www.ipma.ch
10. Página de Agile Alliance: http://www.agilealliance.org
11. Página de Agile-Spain: http://agile-spain.org
12. Página de Scrum Alliance: https://www.scrumalliance.org/
13. Repositorio del Institute of Electrical and Electronics Engineers (IEEE): http://ieeexplore.ieee.org

Anexo: Manifiesto Ágil

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual Respuesta ante el cambio sobre seguir un plan
- Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

Firmado el 17 de febrero de 2001 por *Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas*.

Tras los cuatro valores descritos, los firmantes redactaron los siguientes puntos, como los principios que de ellos se derivan:

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida del progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.

- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se autoorganizan.
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

El Manifiesto Ágil está disponible en http://agilemanifesto.org